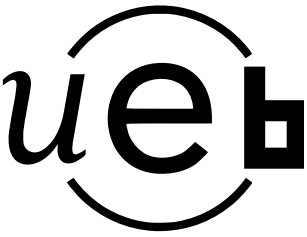
Evolution, testing and configuration of variability systems intensive

José Ángel Galindo Duarte

ANNÉE 2014



THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

En cotutelle Internationale avec
Université de Séville, Espagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : informatique

École doctorale Matisse

présentée par

José Ángel GALINDO DUARTE

préparée à l'unité de recherche IRISA-UMR6074
Institut de Recherche en Informatique et Systèmes Aléatoires
INRIA

Evolution, testing and configuration of variability intensive systems

Thèse soutenue à Séville le 4 mars 2015

devant le jury composé de :

Miguel Toro Bonilla
Professeur de l'université de Séville */ examinateur*

Sergio Segura Rueda
Professeur de l'université de Séville */ examinateur*

Jean-Marc Jézéquel
Professeur de l'université de Rennes 1*/ examinateur*

Alexander Felferning
Professeur de l'université de Graz*/ examinateur*

Juan Manuel Murillo
Professeur de l'université d'Extremadura*/ rapporteur*

EVOLUTION, TESTING AND CONFIGURATION OF VARIABILITY INTENSIVE SYSTEMS

JOSÉ A. GALINDO

ADVISED BY:
DR. DAVID BENAVIDES CUEVAS AND DR.BENOIT BAUDRY

DOUBLE DEGREE INTERNATIONAL DOCTORAL DISSERTATION

First published in January 2014 by
The Department of Computer Languages and Systems
ETSI Informática
Avda. de la Reina Mercedes s/n
Sevilla, 41012. SPAIN

http://www.joseangelgalindo.es/
jagalindo@us.es

Classification (ACM 2012):

Categories and subject descriptors: D.2.13 [Software Engineering]: Reusable Software: Domain Engineering; D.2.1 [Software Engineering]: Requirement/Specifications: Languages,Tools; D.2.5 [Software Engineering]: Testing and Debugging - Testing tools

General Terms: Design, Theory, Algorithms, Performance

Additional Key Words and Phrases: software product lines, software product families, feature models, automated analysis, variability, combinatorial testing, variability intensive systems

Support: This work has been partially supported by the European commission (FEDER), by the Spanish government under TAPAS (TIN2012-32273) project, by the French government under MOTIV project and by the Andalusian government under the Talentia scholarships program and under THEOS (TIC-5906) and COPAS (TIC-1867) projects.

Don David Felipe Benavides Cuevas y Don Benoit Baudry, profesor titular de la Universidad de Sevilla, e investigador principal del grupo de investigación Diverse en INRIA,

HACEN CONSTAR

que Don José Angel Galindo Duarte, Ingeniero en Informática por la Universidad de Sevilla, ha realizado bajo nuestra supervisión el trabajo de investigación titulado

Evolution, testing and configuration of variability intensive systems.

Una vez revisado, autorizamos el comienzo de los trámites para su presentación como tesis doctoral al tribunal que ha de juzgarlo.

Fdo. Dr. David Felipe Benavides Cuevas y Dr. Benoit Baudry
Universidad de Sevilla,
Sevilla, Octubre de 2014

Yo, José Ángel Galindo Duarte, con DNI número 48959770-S,

DECLARO

Ser el autor del trabajo que se presenta en la memoria de esta tesis doctoral que tiene por título:

Evolution, testing and configuration of variability intensive systems.

Lo cual firmo en Sevilla, Octubre de 2014.

Fdo. José A. Galindo

In addition to the committee in charge of evaluating this dissertation and the two supervisors of the thesis, it has been reviewed by the following researchers:

- Dr. Jules White (Vanderbilt University, USA)
- Dr. Mathieu Acher (Université de Rennes 1, France)

10

A Mis abuelos

José Duarte, José Galindo, Lourdes Rodríguez y Valme Vaquero

Contents

List of Figures

List of Tables

Agradecimientos

Finalmente ha llegado la hora de poner punto y seguido a una etapa llena de momentos tan duros como enriquecedores. Estos momentos me han permitido tanto hacer nuevos amigos como echar en falta a otros.

Gracias a mis padres por enseñarme a no rendirme y seguir en los momentos difíciles y a Vanesa por estar siempre conmigo por muchos kilómetros que nos separasen. Nunca olvidaré esa primera despedida en la que os decía un hasta pronto a los tres. Asimismo, dar las gracias a aquellos amigos que hicieron que me sintiese en casa estando lejos. Hamilton, Anna, Joon, Juan, Ania, Ramón, Mauricio, Jesús, Paco y otros tantos que es imposible mencionarlos aquí.

También ha sido muy importante el apoyo de mis tutores David y Benoit para la conclusión de esta tesis. El primero, por enseñarme el oficio y animarme en todo momento. El segundo, por haber estado ahí cuando más hacía falta. También dar especiales gracias a Jules White y a Mathieu Acher por acogerme en sus grupos de trabajo. Finalmente también agradecer el apoyo recibido a los grupos de investigación ISA, Diverse y Magnum por haberme abierto las puertas y puesto a mi disposición todos los recursos que tenían disponibles.

Abstract

One of the key characteristics of software is its ability to be adapted and configured to different scenarios. Recently, software variability has been studied as a first-class concept in different domains ranging from software product lines to pervasive systems. Variability is the ability of a software product to vary depending on different circumstances. Variability intensive systems are those software products where variability management is a core engineering activity. The varying parts of those systems are commonly modeled by using different variability model flavors, being feature modeling one of the most common ones. Feature models were first introduced by Kang et al. back in 1990 and are a compact representation of a set of configurations in a variability intensive system.

The large number of configurations that a feature model can encode makes the manual analysis of feature models an error prone and costly task. Then, computer-aided mechanisms appeared as a solution to extract useful information from feature models. This process of extracting information from feature models is known as *"Automated Analysis of Feature models"* that has been one of the main areas of research in the last years where more than thirty analysis operations have been proposed.

In this dissertation we looked for different tendencies in the automated analysis field and found several research opportunities. Driven by real-world scenarios such as smart phone or video–surveillance domains, we contributed applying, adapting or extending automated analysis operations in variability intensive systems evolution, testing and configuration.

Concretely the main gaps addressed by this thesis are:

Testing. Two different solutions are presented in this context. First, we show how to prune, prioritize and package variability intensive systems configurations by the use of attributed feature models. This solution brings cost and value guided testing to variability intensive systems. Second, we present a hybrid solution to enable the prioritization of pair-wise cov-

ering sets while taking into account multiple stakeholders preferences. Both, approaches are validated in different contexts such as smartphone ecosystems and video-sequences generators.

Evolution. Evolution is intimately related with testing, being one of the main task in the evolution process. Evolution of variability models may span between different years. We propose mechanisms to grant the safeness of the transition of a configuration between different evolution phases. We evaluated this approach within an avionics scenario.

Configuration. We propose a solution to configure diverse variability intensive systems that have been modeled with different variability model languages and have different visibility restrictions. We validated this approach by using the motivating scenario of the Android permission system.

Resumen

Una de las características más importantes del software, es la habilidad de adaptarse a distintos escenarios. Recientemente, la variabilidad del software es estudiada como un elemento esencial en distintos dominios que varían desde las lineas de producto software a sistemas de cloud. Este tipo de sistemas son conocidos como sistemas de alta variabilidad. Los sistemas de alta variabilidad son sistemas software que, debido a su naturaleza, gestionan una gran cantidad de artefactos software variables.

Los modelos de características surgieron para representarlas partes comunes y variables de los sistemas de alta variabilidad. Asimismo, el elevado número de configuraciones existentes en un sistema de alta variabilidad imposibilitan el análisis manual de los mismos. Para salvar este inconveniente, distintos investigadores han propuesto el uso de mecanismos y herramientas informáticas. Esto es conocido como el análisis automático de modelos de características. Recientemente, el análisis se ha aplicado a una variedad de sistemas de alta variabilidad como pueden ser sistemas de computación en la nube o la gestión del kernel de Linux.

En esta tesis, exploramos las diferentes tendencias en el análisis automático de modelos de características. En primer lugar, hemos identificado diversas preguntas de investigación que deben ser abordadas y propuesto diferentes soluciones. Más tarde, hemos validado nuestras soluciones en sistemas de alta variabilidad reales como son el ecosistema Android y los sistemas de generación de vídeo-secuencias.

Concretamente las principales áreas en las que esta tesis se ha enfocado son:

Pruebas: Dos soluciones se han desarrollado en este área. Primero, mostramos como reducir, priorizar y empaquetar conjuntos de pruebas usando modelos de características con atributos de calidad. Después, modelamos las distintas operaciones como un problema de satisfacción de restricciones y validamos usando el ecosistema Android. Segundo,

habilitamos una operación para priorizar la ejecución de conjuntos de configuraciones a la vez que mantenemos una cobertura por pares de las mismas. Esta aportación se desarrolla en el contexto de los sistemas de vídeo-vigilancia. En estos sistemas se necesitan vídeos para proceder a su prueba. En esta tesis proponemos gestionar la variabilidad de cara a optimizar la generación de vídeos de prueba para estos sistemas.

Evolución: Los sistemas de alta variabilidad están sujetos a la evolución debido a la inclusion de nuevos requisitos. Para garantizar la corrección del sistema, es necesario verificar los sistemas de variabilidad en cada fase de la evolución. También es común que haya usuarios de configuraciones procedentes de versiones anteriores que deseen migrar a las más actuales, introduciendo restricciones al migrar. Para gestionar estas transiciones hemos implementado una solución que nos permite optimizar los cambios a realizar en nuestras configuraciones. Esta propuesta se ha evaluado en el contexto de aeronáutica.

Configuración: Finalmente, la complejidad de los sistemas de alta variabilidad imposibilitan la configuración del sistema por una persona. Más aún, una persona no suele tener visibilidad de todo el sistema de alta variabilidad. Para solventar este problema, desarrollamos una solución en la que permitimos la configuración de múltiples modelos de variabilidad. Esta propuesta se evalúa en el contexto de la seguridad en el ecosistema Android.

Résumé

Une particularité importante du logiciel est sa capacité à être adapté et configuré selon différents scénarios. Récemment, la variabilité du logiciel a été étudiée comme un concept de première classe dans différents domaines allant des lignes de produits logiciels aux systèmes ubiquitaires. La variabilité est la capacité d'un produit logiciel à varier en fonction de différentes circonstances. Les systèmes à forte variabilité mettent en jeu des produits logiciels où la gestion de la variabilité est une activité d'ingénierie prédominante. Les diverses parties de ces systèmes sont couramment modélisées en utilisant des formes différentes de "modèle de variabilité", qui est un formalisme de modélisation couramment utilisé. Les modèles de caractéristiques (feature models) ont été introduits par Kang et al. en 1990 et sont une représentation compacte d'un ensemble de configurations pour un système à forte variabilité.

Le grand nombre de configurations qu'un modèle de caractéristiques permet d'encoder rend son analyse manuelle source d'erreur et très couteuse. De fait, les mécanismes assistés par ordinateur sont apparus comme une solution pour extraire des informations utiles à partir de modèles de caractéristiques. Ce processus d'extraction d'information à partir de modèles de caractéristiques est appelé dans la littérature scientifique "analyse automatisée de modèles de caractéristiques" et a été l'un des principaux domaines de recherche ces dernières années. Ainsi, plus de trente opérations d'analyse ont été proposées durant cette période.

Dans cette thèse, nous avons identifié différentes questions ouvertes dans le domaine de l'analyse automatisée et nous avons considéré plusieurs axes de recherche. Poussés par des scénarios du monde réel (e.g., la téléphonie mobile ou la vidéo protection), nous avons contribué à appliquer, adapter ou étendre des opérations d'analyse automatisée pour l'évolution, le test et la configuration de systèmes à forte variabilité.

Concrètement, les principales avancées scientifiques de cette thèse sont les suivantes:

Tests: Deux solutions différentes sont présentées dans ce contexte. Tout d'abord, nous montrons comment découper, prioritiser et preparer des systèmes à forte variabilité par l'utilisation de modèles de caractéristiques avec attributs. Cette solution permet le test de systèmes guidé par le coût et la valeur. Deuxièmement, nous présentons une solution hybride pour permettre la prioritisation des ensembles de paires tout en tenant compte des multiples préférences des parties prenantes. Les deux approches sont validées dans différents contextes: les écosystèmes liés à la téléphone mobile et les générateurs de séquences vidéo.

Les applications de téléphone mobile ("apps") sont généralement exécutées sur une variété de configurations des plateformes sous-jacentes, telles que différentes versions d'Android ou iOS. Chaque configuration exprime différentes caractéristiques de la plateforme mobile, telles que les capacités de résolution écran ou communication réseau (par exemple, 3G, LTE, etc.). Le logiciel doit prendre en considération ces caractéristiques, voire même se configurer différemment selon la plateforme. Par exemple, avec iOS 6, la navigation GPS avec guidage vocal ou visuel est absente sur l'iPhone 3GS mais présente sur l'iPhone 5. En outre, les iPhones 3GS et 4S ont la même taille d'écran, ce qui n'est pas le cas avec l'iPhone 5. L'émulateur Android, qui permet aux développeurs d'émuler les options de configuration d'equipements du monde réel, supporte actuellement 46 caractéristiques différentes de plateforme mobile, qui (en supposant que toutes les fonctionnalités configurables par l'utilisateur pourraient être combinées sans aucune restriction) conduit potentiellement à 2^46 variantes de configuration de plateforme. La grande variabilité qui existe dans l'écosystème Android rend les tests difficiles. Être en mesure de décrire la variabilité de l'écosystème Android comme une ligne de produits logiciels permet aux développeurs d'appliquer des techniques de test existantes et nouvelles pour optimiser les stratégies de test.

Dans un autre contexte, nous considérons les systèmes d'analyse vidéo qui sont omniprésents et cruciaux dans la société moderne. Leurs applications vont de la vidéo protection à la gestion de crise et à l'analyse de foules. Les séquences vidéo sont acquises, traitées et analysées afin de produire une information numérique ou symbolique. L'information correspondante entraîne généralement la notification d'alertes à des observateurs en cas de situations ou d'événements remarquables. Par exemple, un scénario classique dans les catastrophes naturelles est de reconnaître les victimes en utilisant des caméras aéroportées avec l'intention d'agir rapidement sur la base des informations glanées et de fait planifier une stratégie ou tactique de sauvetage.

En fonction de l'objectif de la reconnaissance vidéo, des algorithmes de traitement du signal sont assemblés de différentes manières. En outre, chaque

algorithme est un logiciel complexe, spécialisé dans une tâche spécifique (par exemple, la segmentation et la reconnaissance d'objets). Même pour un travail spécifique, il est difficile de trouver un algorithme qui fonctionnerait de manière efficace et précise dans toutes les situations. Ainsi, l'ingénierie des systèmes d'analyse de séquences vidéo nécessite de choisir et de configurer la bonne combinaison d'algorithmes.

Pour surmonter les limitations précédentes, nous introduisons une approche générative, fondée sur les principes de modélisation et d'analyse de la variabilité. Le but de la démarche est de synthétiser automatiquement une variante d'une séquence vidéo ou d'une configuration Android correspondant à une configuration (c'est-à-dire, une sélection de caractéristiques souhaitées). Par rapport à la pratique actuelle, l'approche vise à fournir une plus grande automatisation, plus de diversification et plus de contrôle lors de l'élaboration de jeu de données de tests.

Evolution: L'évolution est intimement liée aux tests qui est l'une des principales tâches dans le processus d'évolution. L'évolution des modèles de variabilité peut s'étendre sur différentes années. Nous proposons des mécanismes pour s'assurer de la consistance du processus de configuration entre les différentes phases d'évolution. Nous avons évalué cette approche dans un scénario d'avionique.

Lorsque le logiciel évolue, son évolution peut avoir besoin d'être divisée en plusieurs étapes afin de répondre aux contraintes d'évolution. Dans certains cas, les caractéristiques des produits doivent être introduites progressivement dans un laps de temps donné. Par exemple, le Boeing 737, introduit en 1966, a été constamment amélioré et adapté au fil du temps et est toujours actuellement en service. Chaque configuration successive du Boeing 737, qui correspond à une variante, a été développée sur plusieurs années et a incorporé de nouvelles fonctionnalités par rapport à la variante de base de 1966. Par exemple, le développement de la configuration 737-300 de l'avion a commencée en 1979 et son premier vol s'est déroulé en 1984. Le configuration a ajouté une variété de fonctions, comme un système électronique d'instruments de vol. Le 737 a eu de nombreuses configurations successives, tels que le 737-400, 737-500, 737-600, 737-700, 737-800 et le 737-900 qui ont toutes été planifiées et développées pendant de longues périodes.

Il y a un certain nombre de scénarios où l'évolution d'un ensemble de produits peut être effectuée sur plusieurs étapes prédéfinies. Par exemple, quand une nouvelle distribution Linux, comme une nouvelle version d'Ubuntu, est prévue, les développeurs doivent décider l'ensemble des artefacts logiciels qui vont être ajoutés et supprimés dans la prochaine version de la distribution (par exemple, ajouter et supprimer des paquets et modifier leurs dépendances).

En outre, dans d'autres domaines, tels que la construction d'avions ou les centrales nucléaires, la configuration et la mise à niveau des produits sont prévues et analysées des années avant la production réelle (par exemple, les configurations du Boeing 737 ont duré 46 ans). Idéalement, un constructeur aéronautique souhaiterait dériver une séquence de configurations successives qui s'appuient sur les précédentes, comme les variantes du 737 le font, de sorte que plus de fonctionnalités avancées sont incluses chaque année. Un fabricant, cependant, ne peut pas choisir arbitrairement les fonctionnalités à ajouter pour une année donnée. Au lieu de cela, chaque ensemble de fonctionnalités pour une année doit constituer une configuration complète et correcte de la ligne de produit pour éviter la vente d'une configuration défectueuse et non viable.

Nous avons développé un procédé automatisé afin de dériver un ensemble de configurations satisfaisant une série d'exigences sur un ensemble d'étapes de configuration. Notre technique transforme un problème de configuration à plusieurs étapes en un problème de satisfaction de contraintes. Elle utilise ensuite un solveur de contraintes pour générer une série de configurations qui satisfont les contraintes à plusieurs étapes. De plus, elle peut renvoyer soit tous les chemins valides, soit un chemin unique optimisé à partir de la configuration initiale jusqu'à la configuration finale. L'ingénieur des lignes de produit peut ainsi décider quel chemin d'évolution correspond au mieux aux objectifs du projet.

Configuration: Nous proposons une solution pour configurer divers systèmes à forte variabilité qui sont modélisés par différents langages de modèle de variabilité et qui présentent différentes restrictions de visibilité. Nous avons validé cette approche en utilisant le scénario du système d'autorisations d'Android. Les lignes de produits logiciels sont de plus en plus développés au-delà des frontières d'une seule organisation. Par exemple, dans les écosystèmes logiciels, les organisations et les équipes créent des logiciels via un effort de collaboration. La gestion de la variabilité et de la configuration du produit dans de tels contextes doivent concilier les différentes approches de modélisation, les notations et les outils utilisés. En raison des différences significatives dans les pratiques des différents domaines, il est irréaliste de supposer qu'il y aura un jour une approche de modélisation de la variabilité unique et normalisée en dépit des efforts de normalisation en cours. Toutefois, le nombre croissant de "solutions insulaires" à la modélisation de la variabilité et de la configuration de produit restreint la communication et la collaboration entre les ingénieurs de lignes de produits.

Nous proposons Invar, une approche facilitant l'intégration des modèles de variabilité créés via différentes approches de modélisation et potentiellement par des équipes différentes. Dans ce chapitre, nous nous concen-

trons sur les aspects de configuration de produit de notre infrastructure d'intégration. Invar cache les aspects techniques internes de l'utilisation de différents modèles de variabilité aux parties prenantes s'occupant de la configuration. Les outils et formats de données spécifiques utilisés pour définir les modèles de variabilité ne sont pas pertinents pour les utilisateurs finaux qui n'ont besoin de se concentrer que sur les choix de configuration disponibles et leurs implications. Invar unifie les opérations de configuration sur les modèles de variabilité et permet aux modeleurs de choisir librement une représentation de données en accédant aux modèles de variabilité à travers de services Web. Notre approche ne force pas les organisations à intégrer leurs outils de configurations en adaptant la mécanique interne des outils. Au contraire, nous leur permettons de composer leurs mécanismes de configuration en utilisant des définitions de connecteurs et d'interfaces. Nous validons notre approche en intégrant trois dialectes différents de modélisation de la variabilité : modèles de caractéristiques, modèles de variabilité orthogonaux (OVM) et modèles de décisions. Nous montrons aussi comment les scénarios typiques des écosystèmes de logiciels peuvent être supportés par Invar.

Part I

Preface

Chapter 1

Introduction

If such things have not been part of your own experience, you probably won't understand what Bastian did next

Michael Ende, the Neverending Story, Book writer

In this dissertation, we report our work in the evolution, testing and configuration of variability intensive systems. In this chapter we give an overview of the contributions, the research method and publications related to this document.

1.1 Overview

Mass customization [50] is the process of producing a product that meets a wide range of requirement sets while maintaining the low production costs of mass production systems. The key to mass customization is producing a set of varying products that share a common core of components, while allowing specific points of variation in order to meet differing requirements. A more formal definition can be found in Tseng and Jiao [176], mass customization is the process of *"producing goods and services to meet individual customer's needs with near mass production efficiency"*. Figure §1.1 present a motivating scenario of mass customization based in the smart-phone ecosystems, where the application developed for one ecosystem would be able to adapt to a variety of screens and hardware architectures.

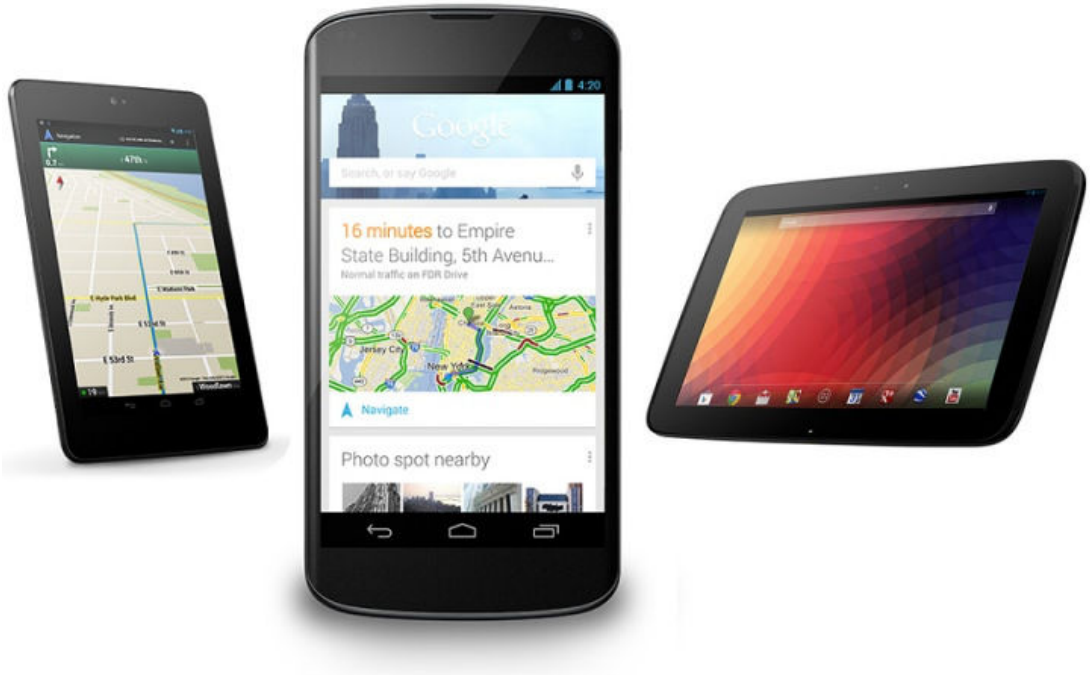


Figure 1.1: *Smartphone masss-customization.*

Mass customization introduce new challenges in software engineering when adapting it to different and diverse scenarios. The varying set of user requirements pushes forward practitioners in the need of managing the variability in a systematic and scalable manner. This points out, that variability

management acts as a proxy between mass customization and software making variability management a crucial task in mass-customization related tasks.

Variability intensive systems are those systems in which, the variability management is a core activity. Different examples can be found in the literature such as cloud-price management systems [73], video-generators [69], debian-based distributions [68], comparison matrices [150] or content management systems like Drupal [171].

To describe the variability existing in variability intensive systems, researchers and practitioners rely on the concept of variability models. Different variability models focus on different aspects of the variability intensive systems. For example, feature models emphasize the description of common and variant functionality while orthogonal variability models describe the variant parts by stacking on a base model. In any case, feature models got hype and become the most used variability model flavor making it to appear several solutions and tools relying in this formalism.

The basic feature model notation[92] constructs are depicted in terms of mandatory, optional and exclusive features as well as propositional constraints over the features. The features are hierarchically organized starting from the high-level concept to more refined and detailed concepts (see Figure §1.2). The essence of a feature model is to characterize a set of valid configurations, where a configuration is defined as the selection of features and attributes values. Propositional constraints and variability information restrict the valid combinations of features authorized in a variability intensive system.

The large number of configurations managed by a variability intensive system enforces the needing of computer-aided mechanisms to avoid the manual analysis which usually is costly and error prone. For example, in Debian based distributions we can find models describing around 28000 variability points [68]. This process of automatically analyze variability intensive systems is known as *Automated analysis of variability models*[19]. This process, starts by taking the model and operational data as input. These artifacts are then, encoded using a logic paradigm. Then, we inquiry the logic paradigm extracting the meaningful information required. Figure §1.2 shows the automated analysis process in terms of variability intensive systems variability management.

Figure §1.3 shows the different software engineering activities where this thesis helps by using automated analysis of variability models. We developed different tools and mechanisms extending existing variability modeling analysis tools and validated in front of real-world variability intensive systems such as smart-phone ecosystems and video-surveillance applications.

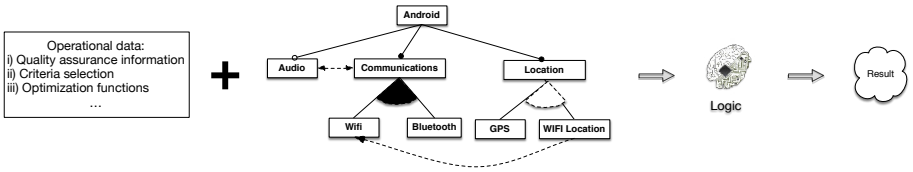


Figure 1.2: *Automated analysis of variability intensive systems feature models.*

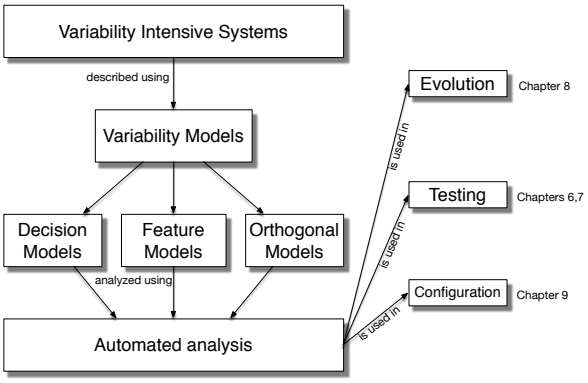


Figure 1.3: *Overview of this thesis scope.*

Concretely, we have coped with i) testing, enabling the cost and value guided testing while relying in quality attributes; ii) evolution, enabling the safe transition between configurations adhered to different versions of the variability intensive system; and iii) configuration, enabling the configuration of distributed and different variability model flavors.

Evolution Evolution is about the introduction of new requirements and functionality's in a software system when the development cycle have started. The evolution and maintenance of variability intensive systems promotes a set of new challenges. For example, to transition users using a product from a previous variability intensive system version to the current one. In this document we propose techniques to grant the safeness in between different versions and validate in front of aeronautics scenario.

Testing While being a crucial part of the evolution, the testing can also occur in far-from-evolution contexts. Moreover, when testing variability inten-

sive systems, the large number of configuration to tests, usually forces practitioners to prune or prioritize the test-cases. In this document we will present different solutions for testing variability intensive systems and validate in front of two real scenarios such as the android ecosystem and the video-sequences generators problems.

Configuration Configuring a variability intensive system is as difficult as to prune the valid configurations from the large set of possible features permutations. Moreover, the distributed and diverse nature of variability intensive systems, makes costly and time consuming this activity. In this document we propose to rely on computer-aided mechanism to assist users when configuring variability intensive systems. Moreover, we validated this in front of the Android permission scenario as well as in front of an enterprise system.

1.2 Research method

The research method used in this research was described by Ida Solheim et al. [84]. This method adapts the traditional research method to the nature of technology research. In technological research, the objective is to create artifacts that are better in some manner than those already developed. This is, that the new artifacts improves the previous in speed, safeness or any other technological characteristic. Once the new artifacts are designed and developed, researchers have to show that the artifacts are complete. Note that this doesn't mean that the artifacts are complete from an end-user perspective but only from the requirements perspective. In other words, check that the artifacts satisfies all requirements. Concretely, the main steps in technology research are:

i **Problem statement:** where researchers look for the potential need of a new technology. This is, where researchers identify the need for new artifacts.

ii **Contribution:** in this step is where the researchers actually develop a solution and create the artifacts supporting the solution.

iii **Validation:** in this final step, researchers verify that all the requirements required to improve the previous solutions and, of course, to grant the solution of the problem have been fulfilled.

Following this research method we started to pose the abstract research questions that would guide our thesis, being it the following one:

How to apply the automated analysis of feature model in variability intensive systems?

To fully understand this question we need to look to the words that provide more meaning to the the question sentence.

Apply Means *"to put to use especially for some practical purpose"*[†1]. Therefore, to apply automated analysis of feature model we need to detect the main points in the area of variability intensive systems were is required.

Automated Means *"to run or operate (something, such as a factory or system) by using machines, computers, etc., instead of people to do the work"* [†2]. In our context, this is to apply computer-aided techniques that help or guide humans.

Analysis Means *"to careful study of something to learn about its parts, what they do, and how they are related to each other"* [†3].

Feature models As explained above, feature models are a common way of representing the commonalities and variability of a software product line.

Variability intensive systems Means those systems that because of their nature needs to cope with a large set of software artifacts

Concretely, the process we followed up in this thesis (which is inspired in [84]) was to i) model the variability existing in this problem; and ii) propose a technique to solve it and evaluate it. To perform such evaluation we have worked with different companies in tandem and resolved real-world problems. Concretely, the main problems we used to validate our research were the ATAACK cloud (funded by the DoD of the USA) and the MOTIV project (funded by the french DGA). In the ATAACK cloud project, we build up a cloud system that enable users to test their Android applications in different systems platforms that vary in different properties such as screen density or input methods. The MOTIV project aims at evaluating computer vision algorithms such as those used for surveillance or rescue operations. A targeted scenario is usually as follows. First, airborne or land-based cameras capture on-the-fly videos. Then, the processing and analysis of video sequences are performed to detect and track, for example, survivors in a natural disaster.

[†1] http://www.merriam-webster.com/dictionary/apply
[†2] http://www.merriam-webster.com/dictionary/automated
[†3] http://www.merriam-webster.com/dictionary/analysis

1.3 Contributions

In this section, we summarize the main contributions of our work. Some of these contributions have been published in different workshops, conferences and journals.

1.3.1 Summary of contributions

This thesis delves in the research relative to software product lines performed in the ISA[†4], Diverse[†5] and [†6] research groups. First, we present the main trends in the automated analysis of feature models. Later, we deep in the different trends were we found research opportunities and show the contributions. Finally, we show how we have validate our contributions by using problems coming from the industry.

The main goal of this thesis document is to provide a set of variability based techniques that help in real scenarios by using feature modeling and artificial intelligence techniques. This is, we applied a divide and conquer approach to answer the main research question. Below, you will find the list of contributions done and questions we answered in this dissertation:

i) **Twenty-five years of feature models automated analysis**

To shell the works done by the feature model automated analysis researchers we performed a mapping study [184] based in Benavides et al. previous work [19].

– *Problem statement:* Previously to this research, Benavides et al. [19] presented a systematic literature review that provides a comprehensive list of the automated analysis of feature models. However, in the last five years, new research trends have emerged.

– *Contribution:* We performed a systematic mapping study to identify the main trends in the area and to position the impact of this thesis inside the automated analysis area.

– *Validation:* In this case, the results of this first contribution have been used to guide the realization of this thesis as well as provided the main

[†4] http://www.isa.us.es
[†5] http://diverse.inrisa.fr
[†6] http://magnum.io

foras where to publish the thesis results. The research questions related to the contribution are:

What is the current status of the automated analysis of feature models?
What are the current trends in the area?
Where does the community publish?

See Chapter §4 for more details about this contribution.

ii) **Testing variability intensive systems: Pruning, prioritizing and packaging target testing platforms**

After determining the current state of the art in feature modeling we identified that there was a lack of testing mechanisms using quality attributes, thus, this was our first contribution.

– *Problem statement:* A daunting challenge is to explore how the cost and value of test cases can be modeled and optimized in order to have more profitable test cases.

– *Contribution:* We worked out an approach that uses automated analysis of feature models to optimize the testing of variability intensive systems. to enable it so we model test value and cost as feature attributes and then we use a constraint satisfaction solver to prune, prioritize and package product line tests complementing prior work in the software product line testing literature.

– *Validation:* We validated our approach in the context of the ATAACK cloud in Virginia Tech. Also, we showed how our approach scales and improves the testing of Android application. Note that, the Android ecosystem because of the diversity it manages, can be considered as a variability intensive system. The research questions related to the contribution are:

How can we improve the testing of variability intensive systems?
How can we select the most profitable set of products in an variability intensive system?

See Chapter §6 for more details about this contribution. This contribution has bee published in the software quality journal (http://link.springer.com/journal/11219).

iii) **Advanced test selection in variability intensive systems: Attributed pair-wise test prioritization.**

When we finished the research of the first three operations we detected that our solution lacked a way of optimizing more complex pruning functions such as the pair-wise [109]. Moreover, we were not able to cope with multi-optimization problems. This is, when users want to optimize more than one function.

– *Problem statement:* A next step when coping with quality attributes in testing is to enable the t-wise generations by considering them. Moreover, there are conflicting optimization functions that require to be satisfied in tandem. In the MOTIV project[†7] we faced the challenge of deriving meaningful configurations that represent video sequences. It is a requirement to provide a good feature and attribute coverage at the same time we optimize certain values that narrow the scope of the videos, thus, being more optimal for certain algorithms and scenarios. This is, video analysis practitioners may be generating images with a good coverage and with a minimum luminosity (simulating videos recovered at night).

– *Contribution:* We developed a hybrid solution that mixes CSP and genetic algorithms allowing to obtain the pair-wise covering set of configurations that optimizes a certain quality attribute. This solution, takes as input a feature model and a set of functions which are used to derive optimal test-suites. Also, this solution acts differently depending if there are more than one testing objective or not.

– *Validation:* We validated our approach in the context of the MOTIV project, a French funded project. In the context of this project we show how with our approach we can derive realistic and meaningful video-sequences for testing videos. Again, we apply the automated analysis of feature models to the video context which can be also considered as a variability intensive system because of the large amount of varying things appearing in a video-sequence.

The research questions related to the contribution are:

How can we manage to select a good covering set that optimizes one or multiple stakeholders criteria?

See Chapter §7 for more details about this contribution. This contribution have been published in the ISSTA'14 conference (http://issta2014.org/).

iv) **Managing feature model drifts: Coping with configuration in evolution scenarios**

[†7] motiv.irisa.fr

As a result of the first contribution we detected a lack of support for configuration in evolutive scenarios. Software evolution is the process of determining how existing software can be adapted to support new customer requirements. Lenhman et al. [99], for example, have explored how software reuse can be employed in software evolution. In particular, their work showed that there is often a set of evolution rules that must be adhered to during the evolution process. In this area we address the configuration plan over multiple steps.

– *Problem statement:* . In software product-lines, software functionality is usually encapsulated in components, enabling the reuse of code between different versions of the software through these components. Because these software components cannot be arbitrarily composed, rules are needed to specify how features can be composed across multiple steps. There are a number of scenarios where the evolution of a set of products may be performed over several predefined steps. For example, when a new Linux distribution such as an Ubuntu release is planned, developers have to decide the set of software artifacts that are going to be added and removed in the next release of the distribution (e.g. add and remove packages and change the dependencies between them).

– *Contribution:* We have developed an automated method for deriving a set of configurations that meet a series of requirements over a span of configuration steps. We call our technique the *MUlti-step Software Configuration probLEm Solver* (MUSCLES). MUSCLES transforms multi-step feature configuration problems into *Constraint Satisfaction Problems* (CSPs) [178]. Once a CSP has been produced for the problem, MUSCLES uses a constraint solver to generate a series of configurations that meet the multi-step constraints. MUSCLES can return either all valid paths or a single optimized path from the initial configuration to the final one and the software product line engineer can decide which evolution path best fits the project's goals.

– *Validation:* We validated the scalability of our approach by using a motivating scenario based on aeronautics. Nevertheless, the example has been provided by some of our colleges that works in the aeronautics sector.

The research questions related to the contribution are:

How can we grant the safeness when deriving a set of configurations in a variability intensive system that spans over multiple configuration steps?

See Chapter §8 for more details about this contribution. This contribution have been reported in the journal of systems and software (http://www.journals.elsevier.com/journal-of-systems-and-software/).

v) **Configuring diverse and distributed variability intensive systems**
The last challenge we addressed in this thesis is the configuration of diverse and multiple variability models that represents one variability intensive system.

– *Problem statement:* In industrial settings, products are rarely developed by one organization alone. Instead, software vendors and suppliers typically maintain their own product lines, which the contributes to a larger (multi) product line or software ecosystem. It is unrealistic to assume that all participating organizations agree on using a specific variability modeling technique—they will rather use different approaches and tools to manage the variability of their systems.

– *Contribution:* We developed an integrative approach that provides a unified perspective to users configuring products in multi-product line environments, regardless of the different modeling methods and tools used internally.

– *Validation:* We validated our approach in the context of the Android security permissions. For doing that, we modeled the different parts of the Android ecosystem as variability models –choosing the most convenient each time– and then pluged in in our tool.

The research questions related to the contribution are:

How can we enable different practitioners to work in tandem over multiple models describing a variability intensive system?

See Chapter §9 for more details about this contribution.

1.3.2 Publications in chronological order

Our research work have followed a frutful path that allowed us to publish our results in important conferences and journals. Beloww is a complete list of publications in chronological order.

[2010] In this year, we started to work doing research in our spare time obtaining some interesting results as an outcome. In this year we were motivated by the applicability of software product lines techniques to far-from-spl variability systems. Concretely, in that period, we focused on the analysis of

Debian variability models which describe the different dependencies among packages.

- **ACoTA'10**. **José A. Galindo**, D. Benavides and S. Segura. Debian Packages Repositories as Software Product Line Models. Towards Automated Analysis. Proceedings of the 1st International Workshop on Automated Configuration and Tailoring of Applications (ACoTA'10 - ASE collocated), Antwerp, Belgium.

[2011] In this year, again in our spare time we obtained some publications that ended by guiding this Ph.D dissertation. The reader will find the list of publications and collaborations we were working out. Concretely, we obtained some publications from previous works related to the Debian variability model research, error detection in feature models and started digging in the configuration of distributed product lines.

- **SPLC'11** Deepak Dhungana, Dominik Seichter, Goetz Botterweck, Rick Rabiser, Paul Grünbacher, David Benavides and **José A. Galindo**. Configuration of Multi Product Lines by Bridging Heterogeneous Variability Modeling Approaches. Software Product Lines - 15th International Conference, SPLC, Munich, Germany.
- **FMSPLE'11 José A.Galindo**, Fabricia Roos-Frantz, Jesús García-Galán and A Ruiz-Cortés. Extracting orthogonal variability models from Debian repositories. Proc. of 2nd International Workshop on Formal Methods and Analysis in Software Product Line Engineering (FMSPLE - SPLC colocated), Munich, Germany.
- **FMSPLE'11** Jesús García-Galán, Pablo Trinidad, **José A.Galindo**, A Ruiz-Cortés. Tool supported error detection and explanations on feature models. Proc. of 2nd International Workshop on Formal Methods and Analysis in Software Product Line Engineering (FMSPLE - SPLC colocated), Munich, Germany.

[2012] In this year, we obtained a Talentia scholarship grant that allowed us to formally do research in a full-time schedule. This scholarships was used to stay one year and a half with Prof. Dr. Jules White in Virginia Tech. We also published some results fruit of the reserch work conducted in a cojoint way with Prof. Dr. David Benavides. The research focus was the study of software evolution, the analysis of diverse variability models, the testing of diverse variability models and variability analysis tools and started working on the ATTACK cloud.

- **SSBSE'12** Roberto E. Lopez-Herrejon, **José A. Galindo**, David Benavides, Sergio Segura, Alexander Egyed. Reverse Engineering Feature Models with Evolutionary Algorithms: An Exploratory Study. 4th Symposium on Search Based Software Engineering (SSBSE), Riva del Garda, Italy.

- **SPLC'12** Fabricia R Frantz, **José A Galindo**, David Benavides, Antonio R Cortés.FaMa-OVM: a tool for the automated analysis of OVMs. Proceedings of the 16th International Software Product Line Conference (SPLC), Salvador, Brazil.

- **VAMOS'12** Sergio Segura, **José A. Galindo**, David Benavides, José Antonio Parejo, Antonio Ruiz Cortés. BeTTy: benchmarking and testing on the automated analysis of feature models.Sixth International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS), Leipzig, Germany.

- **JISBD'12** Sergio Segura, **José A. Galindo**, David Benavides, Jose Antonio Parejo, Antonio Ruiz-Cortés.BeTTy: Un Framework de Pruebas para el Análisis Automático de Modelos de Características. XVII Jornadas de Ingeniería del Software y Bases de Datos (JCIS), Almería, Spain.

- **BOOK CHAPTER** H Turner, J White, J Reed, **José A. Galindo**, A Porter, M Marathe, A Vullikanti, A Gokhale. Building a Cloud-based Mobile Application Testbed.Software Testing in the Cloud: Perspectives on an Emerging Discipline, IGI Global, 2013, 382-403, doi:10.4018/978-1-4666-2536-5.ch018.

[2013] In this year the candidate returned back to Europe and worked within the ISA research group for three months. Later on, the Ph.D candidate went to INRIA, Rennes where the student ended by joining to the cotutele program. The main focus in this year was the evolution, configuration and testing of software systems.

- **ICSR'13** David Benavides, Alexander Felferning, **José A Galindo**, Florian Reinfrank. Automated Analysis in Feature Modelling and Product Configuration. International Conference on Software Reuse (ICSR), Pisa, Italy.

- **VAMOS'13** Deepak Dhungana, Dominik Seichter, Goetz Botterweck, Rick Rabiser, Paul Grünbacher, David Benavides, **José A Galindo**. Integrating heterogeneous variability modeling approaches with Invar. Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems (VAMOS), Pisa , Italy.

- **CW'13** A. Felfernig, D. Benavides,**José A. Galindo**, F. Reinfrank. Towards Anomaly Explanation in Feature Models. Configuration Workshop (CW), Vienna, Austria.

- 

JSS'14 Jules White, **José A. Galindo**, Tripti Saxena, Brian Dougherty, David Benavides, Douglas C. Schmidt. Evolving feature model configurations in software product lines. Journal of Systems and Software; 1.14 Impact Factor.

[2014] In this year, we moved to the testing research when dealing with variability intensive systems and started working on the MOTIV project. Later, we obtained several publications related to that project. Also, we published some papers from previous research.

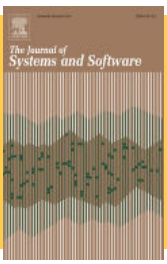
- **SPLC'14** Mathieu Acher, Mauricio Alférez, **José A Galindo**, Pierre Romenteau, Benoit Baudry. ViViD: A Variability-Based Tool for Synthesizing Video Sequences (SPLC), Florence, Italy.

- **ISSTA'14 José A Galindo**, Mauricio Alférez, Mathieu Acher, Benoit Baudry, David Benavides. A Variability-Based Testing Approach for Synthesizing Video Sequences. International Symposium on Software Testing and Analysis (ISSTA), Bay Area, California.

- **VAMOS'14** David Benavides, **José A. Galindo**. Variability management in an unaware software product line company: an experience report. Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS), Nice, France.

- **JISBD'14** Fabricia Roos-Frantz, **José A. Galindo**, David Benavides, Antonio Ruiz Cortés, and Jesús García-Galán. Automated Analysis of Diverse Variability Models with Tool Support. Actas de las X Jornadas de Ciencia e Ingeniería de Servicios (JCIS), Cádiz, Spain.

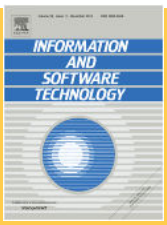
- **JISBD'14 José A. Galindo**, David Benavides, Mauricio Alférez, Mathieu Acher and Benoit Baudry. WindRose: A Cloud-Based Integrated Development Environment for the Automated Analysis of Feature Models. Actas de las X Jornadas de Ciencia e Ingeniería de Servicios (JCIS), Cádiz, Spain.

- 

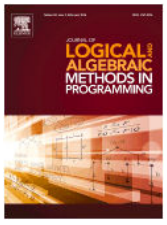
JSS'14 Roberto E. Lopez-Herrejon, Lukas Linsbauer, **José A. Galindo**, José Á. Parejo, David Benavides, Sergio Segura, Alexander Egyed. An Assessment of Search-Based Techniques for Reverse Engineering Feature Models. Journal of Systems and Software; 1.14 Impact Factor (In press).

- 

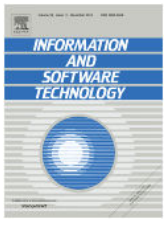
SQJ'14 José A.Galindo, Hamilton Turner, David Benavides and Jules White. Testing variability-intensive systems using automated analysis. An application to Android. Software Quality Journal 2014; 0.8 Impact Factor (In press).

- 

IST'14 José A Galindo, Deepak Dhungana, Goetz Botterweck, Rick Rabiser, Paul Grünbacher, David Benavides. Supporting Distributed Product Configuration by Integrating Heterogeneous Variability Modeling Approaches. Information and Software Technology (In press).

- 

JLAMP'14 Ganesh Khandu Narwane, **José A. Galindo**, Shankara Narayanan Krishna, David Benavides, Jean-Vivien Millo, S Ramesh. Traceability analyses between features and assets in software product lines. Journal of Logical and Algebraic Methods in Programming (Submitted).

- 

IST'14 Mauricio Alferez, **José A. Galindo**, Mathieu Acher, Benoit Baudry, David Benavides. Modeling Variability in the Video Domain: Language and Experience Report. Software: Practice and Experience; 1.14 Impact Factor (Submitted).

In Figure §1.4, we are showing the chronological order of publications. Please note that the Ph.D years are 2012,2013 and 2014. This, also shows the publications depending on the position in the order of authors of each paper and the research subarea.

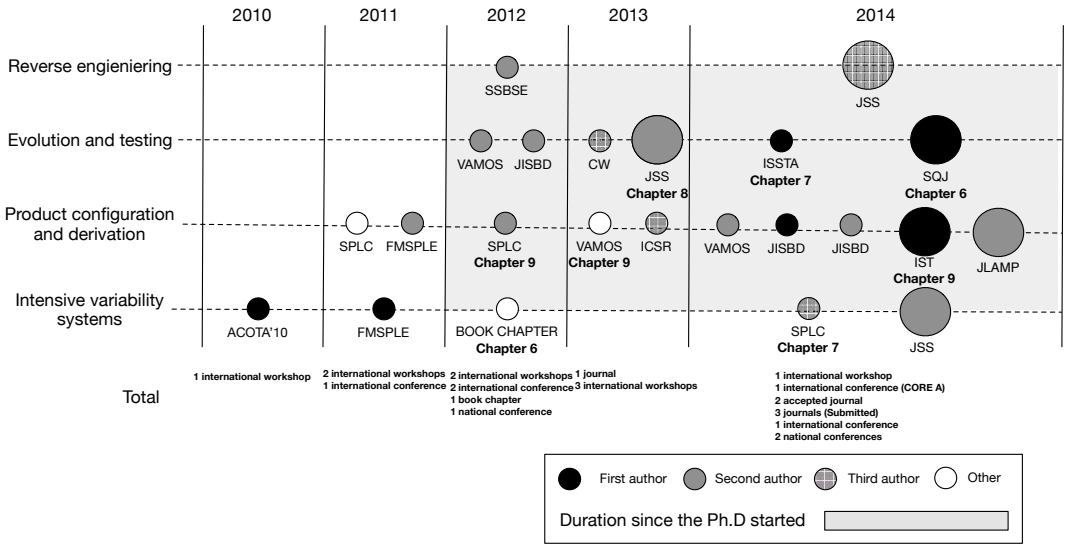


Figure 1.4: *Chronological order of publications.*

1.3.3 Tools

Three different tools were developed during this Ph.D. First, we developed TESALIA [†8] which is a tool to prune, prioritize and package the different products within a product lines. Second, we developed VANE [†9]. VANE is a tool that is able to derive optimal pair-wise depending on different optimization functions. Finally, we developed WindRose [†10], a web based IDE that integrates all the facilities and tools we developed during this research.

1.4 Research internships and collaborations

During the duration of this Ph.D, the research have been done in three different countries and worked with colleges from more than twenty nationalities. Figure §1.5 shows the periods the student spent with diverse hosts and advisers in each country while Table §1.1 shows the affiliation, name of each one and co-authored papers.

[†8] http://tesalia.github.io
[†9] http://vane.github.io
[†10] http://windrose.github.io

	Name	Affiliation	Papers
1.	David Benavides	Universidad de Sevilla	[8, 17, 18, 55, 56, 66, 68–72, 104, 105, 143, 144, 156, 157, 179]
2.	Antonio Ruiz-Cortés	Universidad de Sevilla	[74, 88, 143, 144, 156, 157]
3.	Sergio Segura	Universidad de Sevilla	[68, 104, 105, 156, 157]
4.	José Antonio Parejo	Universidad de Sevilla	[104, 156, 157]
5.	Jesús García Galán	Universidad de Sevilla	[74, 88, 144]
6.	Pablo Trinidad	Universidad de Sevilla	[74]
7.	Mathieu Acher	Université de Rennes 1	[2, 8, 69, 70]
8.	Edward Mauricio Alférez Salinas	INRIA - France	[2, 8, 69, 70]
9.	Benoit Baudry	INRIA - France	[2, 8, 69, 70]
10.	Jean-Vivien Millo	INRIA - France	[118]
11.	Jules White	Virginia Polytechnic Institute and State University	[71, 177]
12.	Brian Dougherty	Virginia Polytechnic Institute and State University	[179]
13.	Hamilton Turner	Virginia Polytechnic Institute and State University	[71, 177]
14.	Jeffrey H. Reed	Virginia Polytechnic Institute and State University	[177]
15.	Madhav Marathe	Virginia Polytechnic Institute and State University	[177]
16.	A Vullikanti	Virginia Polytechnic Institute and State University	[177]
17.	Florian Reinfrank	Graz University of Technology	[17, 66]
18.	Alexander Felfernig	Graz University of Technology	[17, 66]
19.	Fabricia Roos-Frantz	UNIJUI - Brazil	[88, 143]
20.	Dominik Seichter	Irish Software Engineering Research Centre	[55, 56]
21.	Goetz Botterweck	Irish Software Engineering Research Centre	[55, 56, 72]
22.	Tripti Saxena	Vanderbilt University	[179]
23.	Aniruddha Gokhale	Vanderbilt University	[177]
24.	Douglas C. Schmidt	Vanderbilt University	[179]
25.	Paul Grünbacher	Johannes Kepler Universität	[55, 56, 72]
26.	Rick Rabiser	Johannes Kepler University	[55, 56, 72]
27.	Alexander Egyed	Johannes Kepler University	[104, 105]
28.	Roberto E. Lopez-Herrejon	Johannes Kepler University	[104, 105]
29.	Lukas Linsbauer	Johannes Kepler University	[104, 105]
30.	Deepak Dhungana	Siemens	[55, 56, 72]
31.	Adam Porter	University of Maryland, College Park	[177]
32.	Pierre Romenteau	IN-PIXAL	[2]
33.	Ganesh Khandu Narwane	Homi Bhabha National Institute, India	[118]
34.	Shankara Narayanan Krishna	Dep. of Computer Science and Engineering, India	[118]
35.	S Ramesh	Global General Motors R&D - India	[118]

Table 1.1: *List of researchers and institutions the student co-authored a work.*

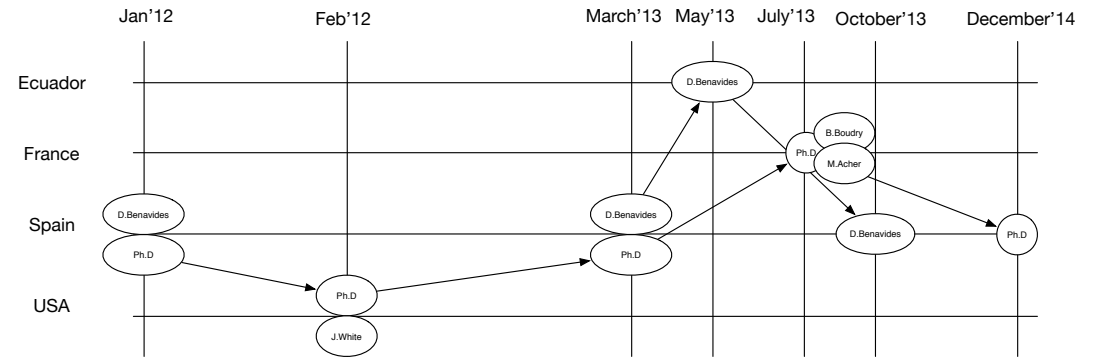


Figure 1.5: *Trips to cope with the Ph.D objectives..*

1.5 Structure of this dissertation

This document is organized as follows:

Part I In the first part of this dissertation we present the main contributions of this research as well as the background and motivating scenarios that pushed forward this work.

Part II In the second part of the dissertation we explore and update the basic background information required to understand the goals of this thesis work.

Part III In this part, we motivate our research work. Concretely we delve in the trends and related work. This is done by performing a mapping study and analyzing the resulting papers. In this mapping scenario, we discovered the main trends in the automated analysis of feature models area while pointing out the foras where the community publishes as well as the main research types researchers focus on.

Part IV In this part we present the main contributions of this thesis. Concretely we will go thought all the different sub areas (testing, evolution and configuration) we researched on. In the testing area, we present two approaches. The first, to prioritize, prune and package products to test based on cost and value functions. Later, we present an approach to prioritize pair-wise covering sets. Later, we dig in the variability intensive systems evolution. Pointedly we propose the use of CSPs to plan the safe evolution of variability intensive systems. Finally, we present our approach to configure a product in those scenarios where there are more than one model describing the variability intensive system.

Part V In this part we present the validation against the different case studies we have worked with. First, we validate TESALIA agains the Android case study, were we model all the configurations existing in the Android emulator. This enabled us to test our Android application in the mobile platform configurations that would provide the most revenue based on market-share data. Later, we validate VANE in the context of an industrial project involving three partners to help in the testing of tracking and movement detection algorithms. We showed how our solution scales in this scenario by generating realistic video-sequences. Later on, we validate our contributions in the area of variability intensive system evolution by using examples from the aeronautics industry. Finally, we validate our approach to configure a diversity of distributed models in

lnvar. This is done by modeling the different artifacts related to the Android permission system and, later, show that users can prevent the executions of applications that use not wanted permissions.

Part VI In this part, we show the different conclusions of this thesis and propose future work to address new and challenging open research questions found during the realization of this dissertation.

Part II

Background

Chapter 2

Automated analysis of variability models

If you look at any leaf on any tree branch, it's similar to but not exactly a repetition of the previous branch. So the new science of complexity or showing how an architecture can be produced just as quickly, cheaply and efficiently by using computer production methods to get the slight variation, the self–similarity.

Charles Jencks, Architecture theorist

Variability intensive systems description are usually encoded by using so called variability models. There are different variability models depending on the focus of the mentioned model. Moreover, there are different techniques to extract information depending on the model flavor. In this Chapter, we present the most common models used in the literature as well as the process of automated analysis and current information extraction techniques.

2.1 Variability models

Variability models are used to represent diverse information existing in variability intensive systems. This information helps practitioners to manage variability intensive systems capabilities. While each variability model flavor focuses on some variability intensive systems aspects, there are some common constructs that worth know. Different variability modeling approaches share common characteristics. A *Variability Model* consists of a set of *Variables* and *Constraints* over these variables. Each variable has a *Type*, for instance, *Boolean*, *Integer*, or *String*. Depending on the particular approach there are different *Types of Constraints*, e.g., "Optional Sub-features" and "Alternative Groups" (e.g., different alternative web access options).

2.1.1 Feature models

Feature models are commonly used to represent the commonality and variability of the products inside a software product line. Feature models were first defined by Kang [92] and have become the "defacto" standard for describing variabilities and commonalities when using software product lines. An example feature model based on the mobile industry is presented in Figure §2.1. A feature model represents all the different products in a software product line. Feature models use different kinds of relationships to define the shared and distinct features between different software products. Usually, two different groups of relationships are defined: i) hierarchical relationships to define the options enabled by a variation point in a product line and ii) cross-tree constraints to define restrictions on features that do not share a common parent in the feature model tree. Different feature model representations can be found in the literature [19]. The most well-known feature model representations are:

Basic Feature Models. Figure §2.1 shows a graphical representation of this feature model representation. The basic feature model defines four kinds of hierarchical relationships:

- **mandatory**: this relationship specifies that when a parent feature is present in the product the child feature also must be present;
- **optional**, this relationship refers to features that can be or not in a product if its parent feature is present;
- **set**, this relationship is between one parent feature and more than one

child features, it implies to have from 1 to any number of descendant features if its parent is present in the product. Special cases of the set relationship are **alternative**, choose only one of the features in the set; and **or**, choose any of the features in the set;

In addition to these hierarchical constraints, three cross-tree constraints are defined:

- **requires** indicates that when the origin feature is present in the product, the destination feature must be also present;
- **excludes** ensures that when the origin feature is present in the product, the destination feature is not in the product and vice-versa.
- **complex** cross–tree constraints are more complex constraints that can be econded as propositional formulas, e.g. "feature a requires b or c".

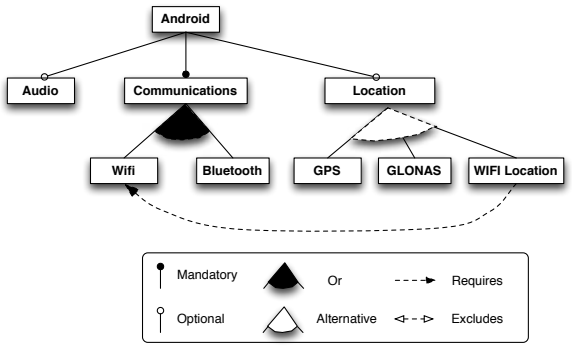


Figure 2.1: *Feature model example from the mobile industry.*

Attributed Feature Models. An attributed feature models is a feature modeling extension that includes attributes on features. Complex constraints are allowed between features and attributes such as "Every attribute called cost must have a value greater than 10." There are a variety of approaches to describe feature model attributes [145], however, most of them share some characteristics. Usually attributes are defined by the use of a name, a domain, a value and a null value. The attribute domain represents the set of values that the attribute can assume. The value, is the value of the attribute when its associated feature is present in the product, the null value is the value of the attribute when its associated feature is not present in the product. In Figure §2.2 an attributed feature model is presented. In the figure, quality attributes are presented showing the accuracy of each location service in a mobile phone.

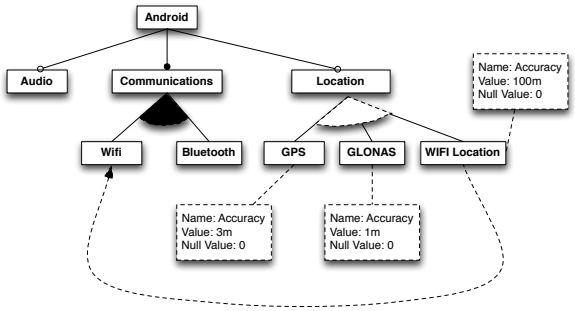


Figure 2.2: *Attributed feature model example.*

2.1.2 Orthogonal variability models (OVM)

OVM is a modeling language for defining the variability of a software product line separately without change the base models (e.g. requirement model, design model). The base models realize the variability defined by the variability elements in the OVM model. In OVM the first-classes are: *variation points (VP)* and *variants*. A variation point documents the aspects that can vary in the product line, which must be chosen by the customer or engineer of the software product line. A variant is related to a variation point and documents how this variation point can vary. OVM is commonly used for documenting software product line variability [135].

Figure §2.3 shows an example of an OVM diagram. In an OVM model a variation point (VP) documents a variable item that can vary from one product to another, and a variant (V) documents how this variation point can vary. All VPs are related to at least one V and each V is related to exactly one VP. Both VPs and Vs can be either optional or mandatory. A mandatory VP must always be bound in all products of the product line. An optional VP can be bound optionally. Binding a VP means making a decision about its variants. A mandatory variant has to be always selected when its parent VP is bound. An optional variant can be optionally selected when its parent VP is bound. In OVM, optional variants may be grouped into alternative choices , i.e., SMS, MMS, or both can be selected as Vs of VP Messaging. This group is associated to a cardinality [min...max]. The cardinality prescribes constraints on how many Vs may be chosen in an alternative choice, at least min and at most max Vs of the group. In OVM, as in feature models, cross-type relationships may be defined between Vs or VPs [113, 135].

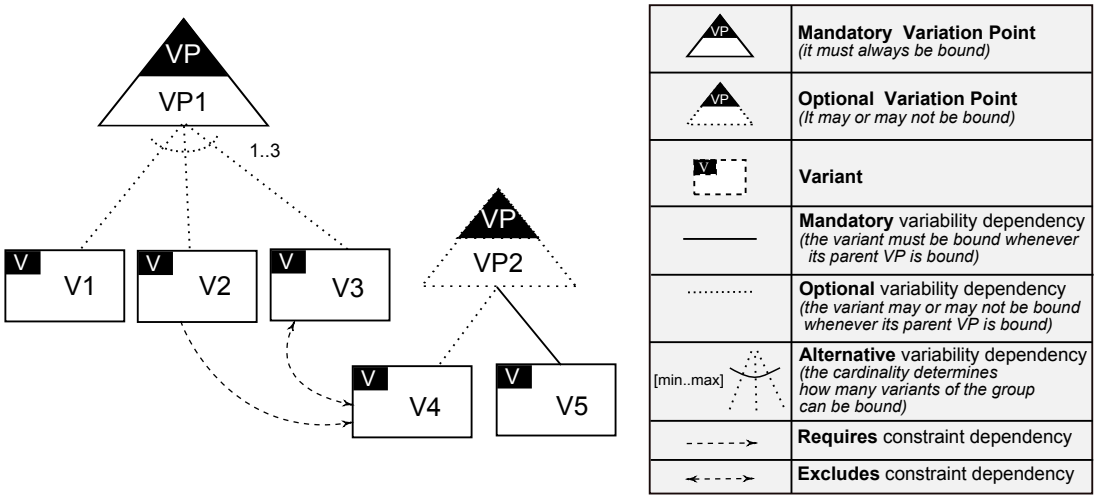


Figure 2.3: *OVM notation.*

Orthogonal variability models have been attributed to support and be interpreted as a set of possible products that can be derived from a variability intensive system. Thus, the OVM metamodel describes the different variability elements and the rules that constraint the combination of these elements in a product line. For, example, Figure §2.4 shows an OVM model containing attributes for a mobile-based ecosystem.

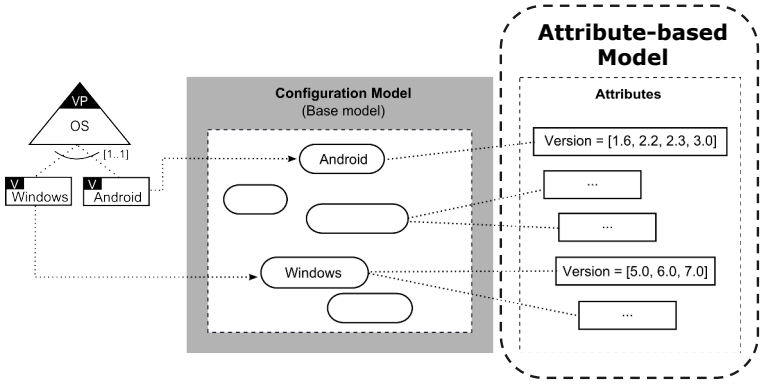


Figure 2.4: *OVM with attributes.*

2.1.3 Decisions models

The family of decision modeling approaches [153] exists nearly as long as feature-oriented modeling. Similar to the role which FODA [90] plays in the context of feature-based variability management most if not all decision modeling approaches have been influenced by the Synthesis method [45]. A decision model describes the differences between products in a product line. It uses a set of multiple related decisions (represented as configuration variables) that need to be made by the user when configuring a product. Decisions are often represented as questions with a defined set of possible answers. Products are derived from a decision model by setting values to the decisions, e.g., by answering questions and following the sequence defined by the decisions' dependencies. The set of values possible for a decision is defined by its data type, e.g., Boolean (users select or unselect an option), Enumeration/Set (users select from a set of possible answers), Number (users enter a numerical value), or String (users set a text as an answer).

A simple schematic for decision models is presented in Figure §2.5.

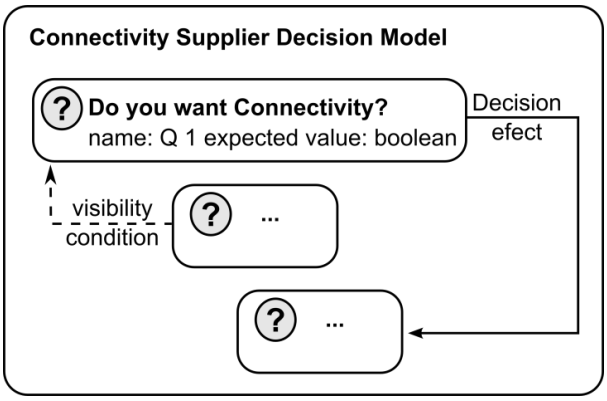


Figure 2.5: *Decision model notation.*

2.2 Automated analysis of variability models

In realistic-sized models is impossible for a human to extract metrics and other information in a manual way. Researchers proposed, thus, the use of computer-aided techniques that extract information from the models. A comprehensive list of the different metrics (a.k.a., operations) existing in the literature can be found in [19].

The automated analysis of feature models consist in a process that takes as input a feature model, then the information contained in the model is translated to a concrete logic paradigm such as CSP or SAT algorithms [96]. Later, we query the logic by adding new constraints and other constructs to the logic. Finally, we extract the metric by analyzing the traceability relation between the logic and the original model constructs. Figure §2.6 shows this process.

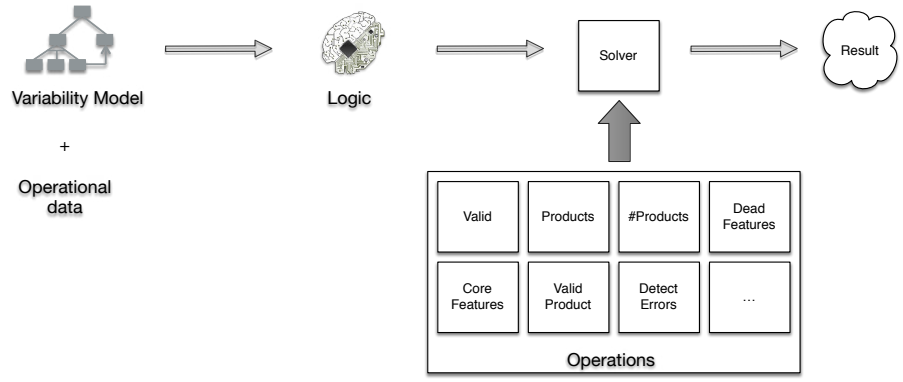


Figure 2.6: *Automated analysis of feature models process .*

Therefore, in this process there are three different variation points. First, the model flavor we use to describe our variability intensive system within operational data such as optimization functions. Second, the logic we use to compute the constraints in the moded. Finally, the operation we execute. In the rest of this section we will go trough the most common logics and operations existing in the literature.

2.2.1 Logics

Constraint satisfaction problem (CSP) → A CSP is a set of variables and a set of constraints governing the allowed values of those variables. A set of values for the variables is called a *labeling*. A valid labeling of the CSP's variables requires that all of the constraints in the CSP are satisfied. For example, in the CSP "$0 < x < y$," where x and y are integer variables, $x = 4, y = 5$ is a valid labeling of the variables. Often, CSPs are used to derive variable labelings that maximize or minimize an objective function. For example, $x = 1, y = 2$ is a solution that minimizes the objective function $x + y$. A key attribute of CSPs is that automated tools, called constraint solvers, can be used to programmatically derive labelings of a set of CSP's variables.

The main benefit of CSP over other approaches is the ability to cope with non boolean variables such as those describing cost and value information.

Boolean satisfiability problem (SAT) → A propositional logic formula is built from i) boolean variables; ii) operators, concretely, the AND, OR and NOT. A formula is said to be satisfiable if it can be made TRUE by assigning appropriate logical values (i.e. TRUE, FALSE) to its variables. The Boolean satisfiability problem (SAT) is, given a formula, to check whether it is satisfiable. The main benefit of this logic flavor is that, because of the nature of the boolean variables they perform fast and quick when coping with feature models. Also, this paradigm is trend and innovative. Currently there are new proposals extending this basic problem, by for example adding different weights to each formulae (a.k.a Qsat/maxsat). Moreover new options to manage non-boolean information are emerging such as pseudo-boolean sat problems.

Binary decision diagram (BDD) → A common way of representing SAT and CSP problems is by using binary diagrams, there is a family of solvers that rely in this representation. The mayor benefit of this solvers is that while they are hight memory users, they can count the number of solutions for a problem in a short time.

2.2.2 Operations

Different operations can be executed by using automated analysis of variability models. Each operation takes a model as input and provides a response as result. More details about analysis operations can be found in [19]. Next we summarize some of the most common analysis operations and exemplify them by using the Android feature model presented in Figure §2.1:

- *Void model*: checks whether a model is void or not, i.e. if it represents at least one valid product. A model may become void due to the wrong usage of constraints. Figure §2.7 shows a void model caused by the inclusion of the cross-tree relationship between "Communication" and "Location". Note that this restriction makes unfeasible the other cross-tree relationship.
- *All Products*: takes as input a model and returns all the valid products represented by this model. Table §2.1 lists the valid products in the model presented in Figure §2.1.
- *#Products*: returns the total number of valid products represented by the model received as input. For example for the model in Figure §2.1 there are 14 different products.

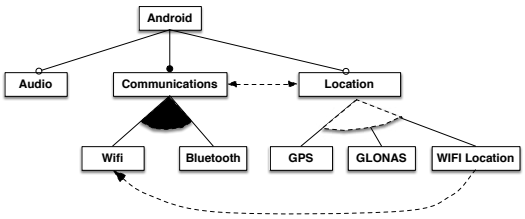


Figure 2.7: *Sample void model.*

Name	Android	Audio	Communication	Wifi	Bluetooth	Location	Wifi Location	GLONAS	GPS
P1	✓		✓	✓					
P2	✓	✓	✓	✓					
P3	✓		✓		✓				
P4	✓	✓	✓		✓				
P5	✓		✓	✓		✓	✓		
P6	✓	✓	✓	✓		✓	✓		
P7	✓		✓		✓	✓		✓	
P8	✓		✓	✓		✓		✓	
P9	✓	✓	✓	✓		✓		✓	
P10	✓	✓	✓		✓	✓		✓	
P11	✓		✓	✓		✓			✓
P12	✓	✓	✓	✓		✓			✓
P13	✓		✓		✓	✓			✓
P14	✓	✓	✓		✓	✓			✓

Table 2.1: *Products result for model in Figure §2.1.*

- *Valid Product*: takes a model and a product (set of variability elements) as input and returns a value that determines whether the input product belongs to the set of products represented by the model or not. For example, if we input the product P14 this operation will return true. However if we input the product {Android and Communication} this will returns false.

- *Valid Partial Configuration*: takes a model and a partial configuration as input and returns a value informing whether the partial configuration is valid or not, i.e. a partial configuration is valid if it does not include any contradiction. This operation will returns valid for the partial configuration {Android and Communication}, however, it will not do it for

the configuration {Android, Wifi location, Bluetooth} as there is a non satisfied constraint in between Wifi location and Wifi.

- *Dead Node*: returns a set of dead nodes (if any), i.e. those that cannot appear in any of the valid products represented by the model. Dead nodes are caused by the wrong usage of constraint dependencies. For example in Figure §2.8 the Audio Feature is a dead node because it is not present in any product.

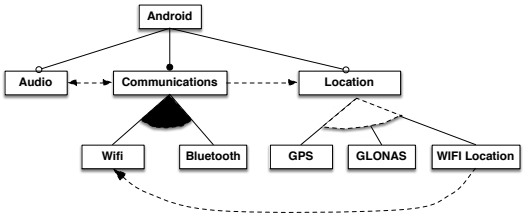


Figure 2.8: *Sample void model.*

- *Commonality*: takes a model and a configuration as inputs and returns the percentage of valid products including the input configuration. For example, the commonality for feature Audio is 1/2.

- *Optimization*: takes as input a model and a function to optimize, returning the configuration that maximizes or minimizes the function. For example, in the model, we can maximize the number of features present in a configuration, this would return all products containing six elements. Also, this would returns only the P1 when minimizing.

2.3 Summary

In this section we presented the main and most common variability modeling flavors such as feature models, orthogonal variability models and decision models. However, is it worth mentioning that there are other model flavors such as Debian variability models [68] and sub-types of variability models (see CVL[169] or Clafer[10]), that mix constructs from the UML world with feature modeling. We also presented the different logics and operations we used as based in this dissertation.

Chapter 3

Evolution, testing and configuration

I have called this principle, by which each slight variation, if useful, is preserved, by the term of Natural Selection.

Charles Darwin, Naturalist and geologist

In software engineering there are different activities that are critical when creating a software system. Moreover, when coping with variability intensive systems, these tasks tends to get more complex because of the large number of products evolved, tested and configured at the same time.

In this chapter we will go through the different activities where we found the challenges addressed in this thesis.

3.1 Evolution

Software evolution refers to accomplish with new software requirements. Those requirements are coming from the aim to satisfy upcoming needing of customers or to solve existing problems with the actual state of the software. Traditionally the terms software maintenance and software evolution have been used as synonyms. Was Lehman [100] who described the software evolution as *"The software evolution phenomenon manifests itself in the common need, for continuing maintenance and periodic upgrades of software used in real world applications"*. He stated that a software cannot be maintained as a mechanical goods because its immateriality. Software, don't suffer from external wear by use. Thus, there isn't any needing to change or replace software parts. The no material nature of the software makes required to perform an evolution process instead of maintaining it. Depending on the project cycle of live the new requirement to be included when evolving can be planned or not. According to the SWEBOK report [1], and shown in Figure §3.1, there are five different steps when coping with software evolution:

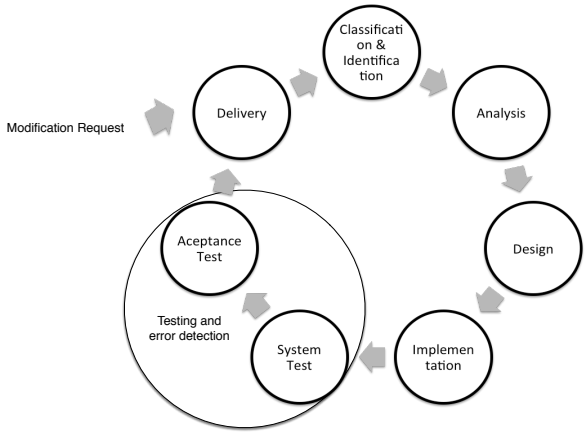


Figure 3.1: *Evolution process.*

Classification and identification. In this step of the evolution process a set of different requirements or changes proposals are used as input. The objective of this step is to classify and prioritize every new requirement. The

classification of the upcoming requirements define the different sub-types of evolution/maintenance defined in the literature [1] i) corrective, that groups the tasks related to restore the system to an operational condition. That, include bug and errors correction; ii) adaptive, that groups the tasks related to cope with the changes in the software environment; iii) perfective, that deal with the introduction of new functionality into the software and iv) preventive, that aims to increase the stability of the software by preventing hidden or latent faults.

Analysis. The aim of this stage is to quantify the impact of every new accepted requirement. Determining the impact of the proposed changes allows developers to determine the associated cost to accomplish a new requirement and then being able to determine if the change worth the investment.

Design. In this stage is were the required modifications for the upcoming requirements are planned. The planned modifications also include the changes required to be done in documentation, software configuration management systems (SCM) and any other artifact related to the software that is going to be evolved.

Implementation. Every artifact affected by the requirement acceptation are modified in this stage. That implies, coding, rewriting documentation or any other artifact identified for change in the design stage.

System Test. In this stage a set of tests are executed on the evolved version of the software. The aim of those tests is to prevent the inclusion of bugs and errors in other artifacts not involved in the evolution process.

Acceptance Test. In this stage a set of tests are executed to verify if the requirement proposed in the classification and identification stage is correct and valid.

Delivery. Finally in this stage, every deliverable of the new release of out product is provided to the owners of the software, it also involves the merging of SCM repositories or any other action pending.

3.1.1 Software product line evolution

The evolution of software product lines enable the safe evolution of software product lines projects. Svahnberg and Wu [168, 186] documented real examples from the industry when evolving software projects. A product line describes a set of deliverable products that shares common assets but designed and maintained as one product. The treatment of software product lines evo-

lution can be archived from two different points of view, i) evolve one of the products contained in a product line; ii) evolve all products at the same time. An example of SPL evolution is the introduction of a new software product in a product line. In this case, companies would need a standardized way to manage the addition of the new product to the product line.

Software products lines are meant to enable the deployment of a set of different products. This characteristic of software product lines makes them a natural step when evolving traditional software projects. Bosh et al. [30] presented six different maturity levels and how the architecture design impacts the profitability of the software product.One existing difference between software product lines and traditional software product is the division of the software functionality into different artifacts and components. The hierarchical division of functionality when referring to software product lines makes errors appears on single artifacts or in the common parts that provides cohesion to the SPL.

3.1.2 Feature model evolution

Open source projects have been dealing with evolution problems by using variability management mechanisms several years ago [106]. Those works represent a guidelines for feature model evolution. However, there are more concrete proposals to cope with feature model evolution such as the EvoPL[134]. The EvoPL enables the transition from requirements to concrete features, this is achieved by adding evolution related information to the model. Later, EvoPL groups features in *EvoFragments*. Thus, a new feature model flavor is defined in base of those fragments and the operators required to enable the evolution. It is also noticeable the Peng et al.[129] contribution, which describes a real case of an evolved feature model showing how the requirements affecting the model.

3.2 Testing of variability intensive systems

Variability intensive systems testing represents a new challenge for software testing researchers [137]. When testing software product lines, each product shares some common functionality with one or more other products, while differing in at least one feature. Software product lines add testing complexity because they require testing a set of products rather than a single product. These products, however, share common functionality or artifacts,

enabling the reuse of some tests across the entire product line.

According to Pohl et al. [137], varying strategies can be used to test software product line products. These testing strategies can be summarized as follows: i) testing product by product, this testing strategy tries to test all products one by one, as if they were not part of a product line. With this strategy the test process covers all possible interactions between features but grows exponentially in cost as a function of the number of features in the product line; ii) incremental testing, this strategy starts by testing the first developed product and creates new unit tests for each new feature added. Using this strategy, the commonalities of a software product line are exploited to reduce testing effort. However, when a new feature is introduced, all the interactions between the new feature and the old ones should also be tested, thus, presenting problems for large software product lines; and iii) Reusable asset instantiation, which relies on data captured in the domain analysis stage of product line creation. With this data, a set of abstract test cases are developed considering all features of the product line. These abstract tests cases are mapped to concrete requirements in the application engineering stage. These testing strategies are designed to reduce the software product line combinatorial explosion in testing cost as a function of the feature count.

3.2.1 Combinatorial testing.

Combinatorial interaction testing [121] exploits the commonalities of a software product to reduce the cost incurred when testing. This approach has been applied to the testing of software product lines because of the functionality encapsulation used in software product line engineering. Concretely t-wise approaches [79, 80, 87, 98, 125, 130, 131] have been used to apply combinatorial testing to software product lines. For example, when using t-wise coverage, at least every feature is covered t times. Combinatorial testing has been proven to produce good results in empirical studies [49, 164], mainly because points of interaction between software artifacts have been proven [95] to be key sources of errors. Nevertheless, there is still consensus that concrete techniques are needed in software product line engineering to aid in reducing testing cost while maximizing the test value because of the large number of potential products that must be tested. Different strategies can be used when applying combinatorial testing on variability intensive systems. Figure §3.2 shows the idea behind this operation.

- **Prune.** → In short, this is to select a subset of products that accomplish some criteria. A very common criteria is to select a set of configurations

that covers t interactions in between features. This criteria is commonly known as t-wise criteria.

- **Prioritize.** → This operation objective is to establish an order when executing the test. Therefore, if a time constraint is introduced in any stage of the development, at least, the most critical test can be executed.

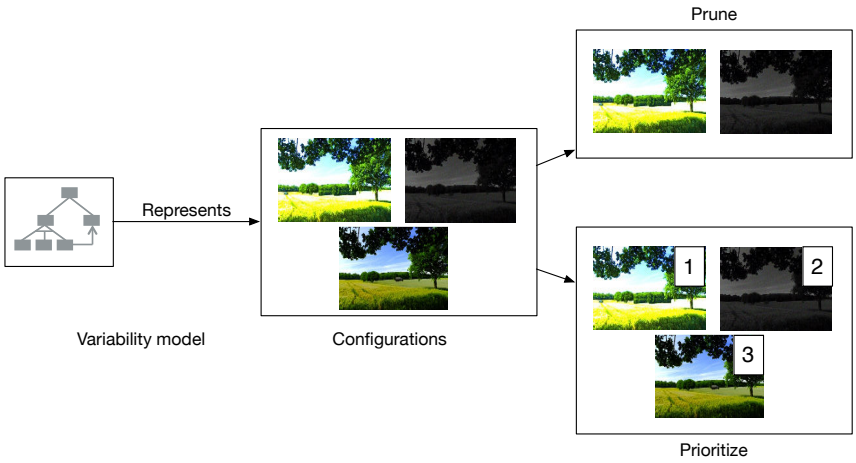


Figure 3.2: *Testing operations overview.*

3.2.2 Cost and value guided testing.

Return on Investment is a metric used in economics that measures the profit generated from an investment. Concretely:

$$\mathrm{ROI} = (\mathrm{benefit} - \mathrm{cost})/\mathrm{cost}$$

This formula shows that when performing an economic activity such as software development, cost and value (benefit) are the variables that should be used as guides. There are several approaches [26] that try to reduce testing cost. Also in software product line testing, combinatorial testing is used to reduce cost [131]. The second variable to be focused on when performing testing is the value [27] of a test. Every time that we execute a test we obtain more confidence that our application is error free. For example in [165], we can see a more detailed explanation of the importance of the value in software development. Also in [167], Srikanth et al, present mechanism to perform value-guided test prioritization.

3.3 Configuration

Configuring is to put together all the different features in a product line or a variability intensive system in such as way that all the constraints within the model are being satisfied. This is, a configuration is composed by a set of features and attributes that are differentiators from other configurations and are not breaking any constrains existing in the model. This selection can be done manually or automatically thus, preventing to break any constraint. The programs that aims to help users when configuring are known as configurations and have been widely used in the automotive industry [†1].

The configuration of feature models can be defined as the process of selecting and deselecting features in a feature model until reaching a full configuration, i.e. a configuration where no additional decision on the feature model needs to be made to have all the information to configure a given software product of the software product line. The configuration of feature models is no more than an analysis operation where the input is a feature model with a set of decisions on the state of a given set of features (a feature can be selected, deselected or undecided) and the output is the feature model together with the new states of the features[175].

Product configuration is an independent area of research from software product line engineering that has a long history as an application of Artificial Intelligence technologies [67, 114, 149]. The first paper on product configuration was published back in 1978 [102]. Similar to feature model based configuration, product configuration can be interpreted as the process of partially or completely instantiating component types and related attributes with concrete components and attribute values [149] in a way that preserves the consistency with a predefined set of constraints (restrictions). Configuration technologies are typically applied in complex product domains such as telecommunication [67] and automotive [89].

An important remark to this definition is that nowadays, the industry is moving towards to open configuration ecosystems where not only one practitioner configures a model. This situation is presented in Figure §3.3 where there are different providers that configure the different parts of the product line. For example, the Messaging feature is configured by providers using ovm while the media feature is configured by a different practitioner using feature models.

[†1] http://app.volkswagen.fr/ihdcc/fr/configurator.html

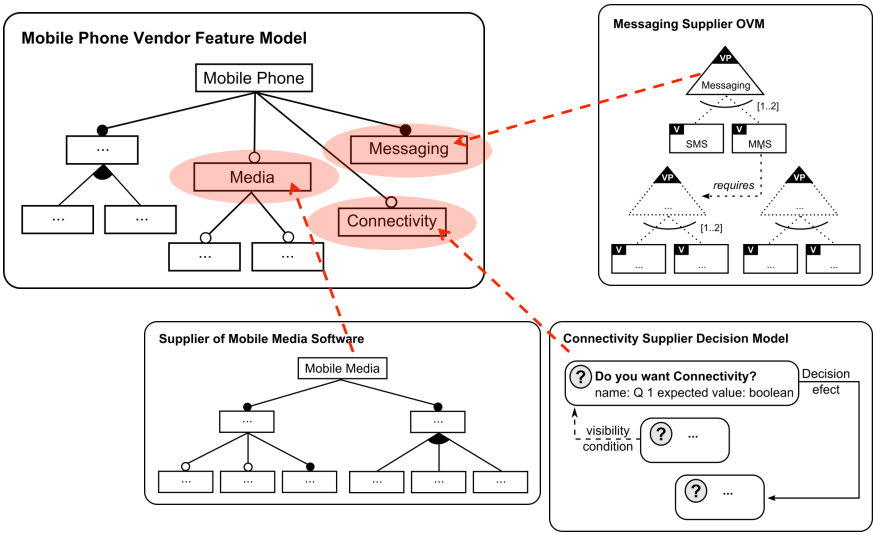


Figure 3.3: *Multi-product line configuration.*

3.4 Summary

In this section we have presented the main concepts and software engineering activities were this thesis delves into. Concretely we explored the evolution, configuration and testing areas.

Part III

Motivation

Chapter 4

25 years of automated analysis of feature models.

Donde muchas cosas hay, de todas hay: buenas y malas, grandes y chicas como en botica.

Dicho popular, Andalusian people

Feature models were invented by Kang et al. 25 years ago. From the very beginning, the automated analysis of feature models was observed as an interested and challenging activity. In 2010 there was a survey on the proposals of feature model analysis after 20 years. In this chapter, we provide an overview of the evolution after five years of this field by performing a systematic mapping. Specially, we overview the applications of the automated analysis that range from cloud computing configurations to mobile phone testing which is beyond the boundaries of software product lines. We also suggest some opportunities for research and applications in the future as well as synergies with other areas such as product configuration. Also, we have detected that the evolution, testing and configuration of feature models are current trends in the feature model research area.

4.1 Introduction

Previously to this research, Benavides et al. [19] presented a systematic literature review that provides a comprehensive list of the automated analysis operations for feature models. However, in the last five years, new research trends have emerged. In this chapter, we present a systematic mapping study to identify the main trends in the area and to position the impact of this thesis inside the automated analysis area.

Different questions remain open at the time of writing this thesis. In research the time passes fast and this made us question what new things appeared after the publication of Benavides et al. [19], thus, we performed a systematic mapping study to find the new mechanisms and tools appeared since the publication.

With this mapping study we want to answer the following research questions:

4.1.1 Research questions

The goal of this work is to provide an overview of the last available research in the area of the Automated Analysis of feature models and thus, identify the main current trends in the area.

- *RQ1. Which challenges are calling researchers attention?* In the last five years, researchers have focused in a diversity of challenges in the automated analysis of feature models area. We want to obtain an overview of the main trends and determine the current direction of the works in the area.
- *RQ2. Which foras are the main targets for publishing research in the area?* Traditionally in the area, researchers tended to publish research on automated analysis in different foras such as SPLC[†1] and ASE[†2]. We are willing to know which foras are the most interesting for in-the-area researchers.
- *RQ3. Which research type facets (e.g., validation papers, philosophical papers) are being left without the researchers attention?* Due the maturity of the area some areas are more likely to lose researchers attraction.

[†1] www.splc.com
[†2] www.ase.com

We want to know which research facets are loosing interest and try to discern why.

4.1.2 Systematic mapping

To get an overview of the existing trends in the automated analysis of feature models we conducted a systematic mapping study considering the papers published in the last five years. Systematic mapping studies are an alternative to systematic literature reviews where the main interest is to determine groups of papers and their relevance. Petersen et al. [132] detailed the process of build systematic mappings and compared them with systematic literature reviews.

The main steps in the process to create a systematic mapping study are: i) defining the research questions to be answered by the study; ii) screening of the papers to be considered; iii) define a classification scheme; and iv) mapping the publications and obtain the conclusions.

In this chapter, we followed up a four step process to obtain the papers considered for the mapping. First, we searched papers citing Benavides et al.[19] previous work since its publication through Google Scholar and Scopus databases. Second, we extended the collection of papers to be considered by regarding the references and adding relevant research. Third, we discarded all non-peer reviewed material such as technical reports and thesis documents. Forth, we removed the conference version of extended papers that were submitted to journals. This is, if a work has been extended and published into a journal, we only take into account the journal version. However, to prevent a bias for the RQ2, we annotated the journal as published in both foras, the journal and the conference.

In the First step, we picked up the papers citing [19] from Google Scholar and Scopus databases. The scholar database returned up to 40 pages of results which does 400 papers in total. Also, we obtained 228 papers from the Scopus database. Because of the unmanageability of this number of papers we only took the 200 first most cited papers from each database. Later, we merged the 200 papers coming from each database and removed the duplicated ones getting 278.

Finally, we performed a search in both databases of the string search *"feature model"* AND (*"reasoning"* OR *"analysis"* OR *"automated"* OR *"analyses"*). Only results from 2010 to 2015 has been considered. Also, the query results were ordered by number of citations. From these queries we took the first 200 results of each database and merged with the results of the second

step. The aim of this last step is to reduce the bias of only considering citations to [19]. Note that, as we were looking forward detecting the different trends in the automated analysis of feature model area, the more related papers we overview the more insights we can find. Finally, from the total of 278 plus the the 400 from the search we removed the duplicates and the non-peer reviewed. This, resulted in considering a total of 250 papers (see Figure §4.1).

RQ1 can be answered by analyzing the abstracts and introduction of the selected papers. Note that this is also part of the keywording process required to do the mapping (see Section §4.2 for more details). RQ2 and RQ3 will be answered by actually executing the mapping, thus, determining the foras where the research has been published and the publication type published.

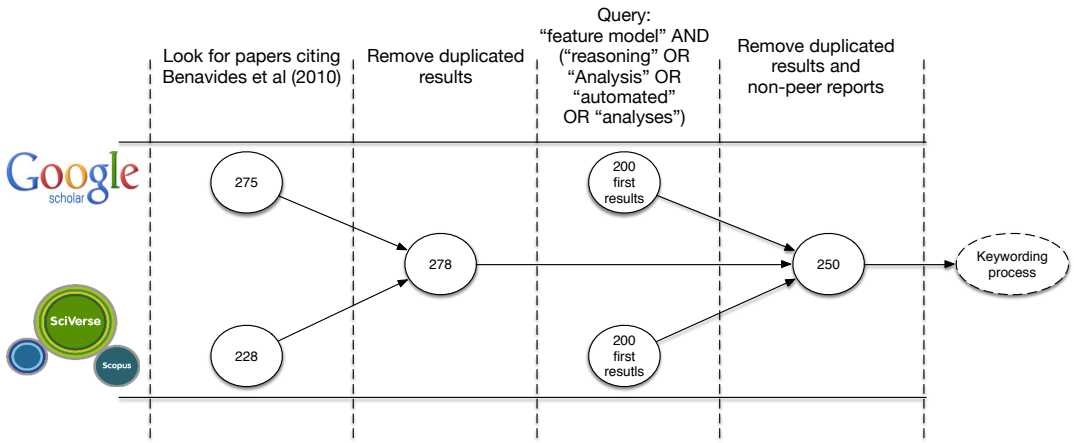


Figure 4.1: *Search citing publications process.*

4.2 Classification scheme

Publications are classified in three dimensions: i)the research focus; ii) the type of contribution; and iii) the research type. We defined these dimensions by following up the process described by Petersen et al.[132]. Petersen et al, propose to use a key-wording based method to define the research focus to group the papers. This process is done in two steps. First, researchers read the abstracts in the collection to review and identify the keywords and concepts that reflects the paper contribution – if the abstracts is not enough then, researchers may need to take a look to the introduction. Finally, by knowing

the main paper contributions, researchers define the set of categories to do the mapping.

We followed up this method to define the different research foci. However, because of the large number of papers that were selected in the screening process described in Figure §4.1, we tried to group the papers in as less groups possible. Thus, to detect the main trends in the automated analysis of feature models. The foci we found were: i) Reverse engineering, these papers describe different techniques to build up feature models from a diversity of product descriptions such as PCMs [150] or lists of products; ii) variability intensive systems analysis, these papers focus on applying automated analysis techniques into systems with a high variability requirements such as cloud service providers [73] or the linux kernel [160]; iii) Variability languages, these papers focus on properly describing the variability to perform further analysis and extraction of relevant information; iv) Multi-product lines automated analysis, these papers focus on the analysis when variability is not described in a single model but in several. This includes for example, merging and slicing operations between models and; v) Software product line testing, these papers focus on the use of automated analysis techniques to prune, prioritize or package the set of products within a feature models.

Different kinds of publications were also taken into account for this mapping study. The different publication types were selected based in the proposal of Wieringa et al.[184] and encouraged in [132]. Concretely: i) Validation research, this research type focus on novel techniques not yet implemented nor validated; ii) Evaluation research, implemented techniques with validation and conclusions; iii) Solution proposal, in these papers authors proposes solution to problems but relying in existing techniques – even improving them; iv)Philosophical papers, these papers try to help structuring the area (e.g., taxonomies); v) Opinion papers, groups papers showing the author opinions over a concrete technique but not relying on methodologies or related work; and vi) Experience papers, papers explaining industrial or personal experiences in the field.

The contribution type is the last dimension to consider in this mapping. Again, we base our selection of contribution types in the Wieringa et al.[184] proposal. Concretely, we chose: Tool, Method, Model, Metric, and Processes. Note that in this case we did not considered any open items category. Tools, refers to any kind of tool demo paper we found but is not limited to tool demos, for example, papers with its main focus on a tool prototype are also categorized as tools. Methods, groups the papers telling about methods to apply SPL techniques. Model, focus more on the modeling tasks required when applying SPL engineering. Metric, categorizes those papers that retrieve metrics

from variability models or other artifacts existing in the SPL. Finally, the processes category encloses the research that propose processes to achieve different tasks in SPL.

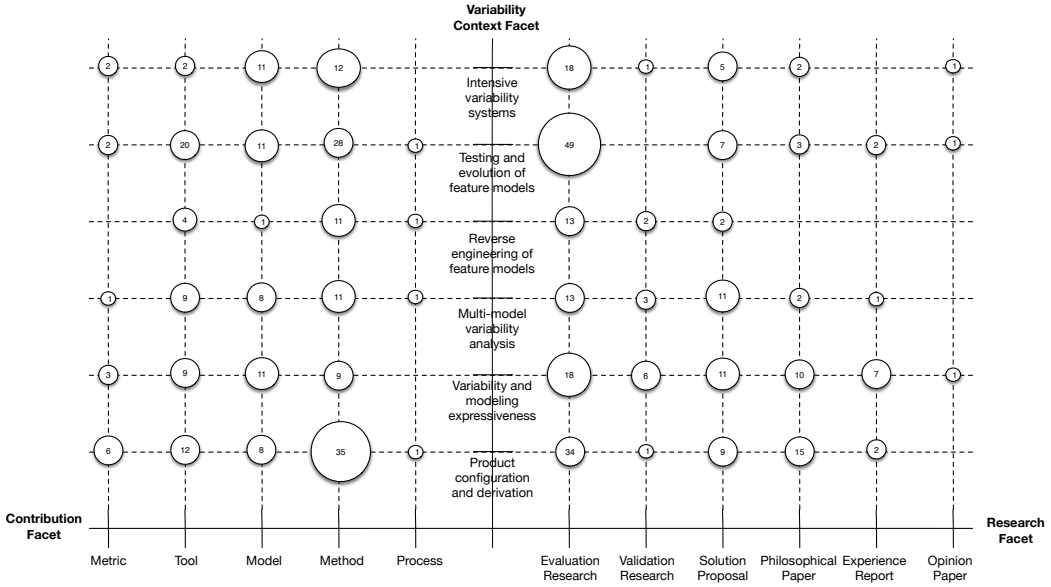


Figure 4.2: *Visualization of the systematic map.*

4.3 Research focus

We read them and extracted the main lines and trends in the research area. Later we briefly present a tour throughout the main trends presenting what haven been done in the last five years. Figure §4.2 shows the distribution of each research focus compared with the contribution facet and the research facet. Moreover, Figure §4.3 shows the percentage of papers per context facet.

4.3.1 Variability intensive systems

The AAFM was initially developed only thinking in software product lines. However, it has ended by being used for a widespread number of applications. Currently, the scenarios were AAFM is used range from variabil-

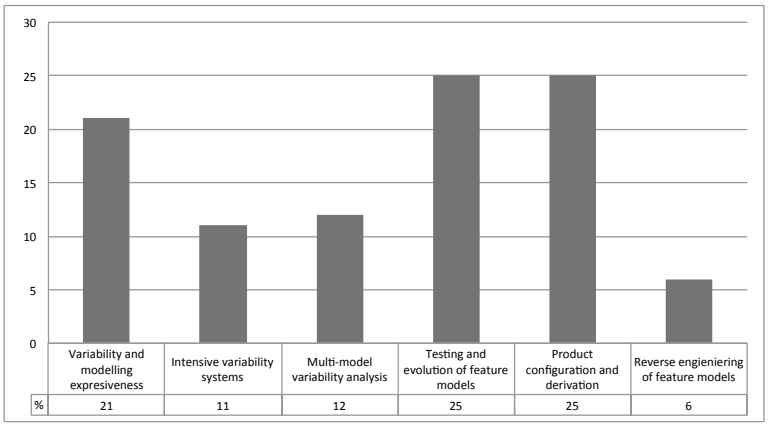


Figure 4.3: *Distribution in percentage of research focus.*

ity analysis in other domains such as BPMN analysis to mobile apps testing, video sequences generation and web services dynamic configuration.

Different variability models are now benefiting from automated analysis techniques to extract information in a look alike to FM manner. While diverse models such as OVM[145] and DOPLER[54] models had automated analysis support in 2010, now, there are far-from-SPL models such as BPMNs[120], that use similar techniques to extract information and metrics. For example, AAFM is used to detect erroneous paths not only in BPMN but also in SAAS[97, 154] applications to grant the satisfiability of contracts.

Further beyond the variability models analysis, researchers looked in the world of cloud computing to make applications running on cloud services more profitable by using AAFM. Cloud providers prices variability have been encoded using attributed feature models [73]. This enabled, for example, to extract useful information rapidly adapting the number and kind of rented cloud instances depending on the network demand.

Also, mobile phone industry is relying on feature model automated techniques. Different researchers modeled the variability present in the Android operating system using feature models. This permit, to optimize mobile application testing by only emulating smartphones with high-market impact [177] and to adapt dynamically Android applications to different situations [127].

Moreover, there are applications of AAFM really away from software product lines such as the embodied ubiquitous learning games, where researchers have managed to succeed applying variability techniques [191]. For exam-

ple, modeling the educational and architectural features of those systems to manage the variants that better adapt to certain student requisites.

4.3.2 Testing and evolution of feature models

Large number of product are encoded in a SPL, making the testing process expensive and tedious. Two main approaches have been followed to reduce testing costs. First, combinatorial testing and more concretely T-wise methods to narrow the number of products to test (see Chapter §6 for more information). Later, test prioritization to grant the execution of critical tests if a time constraint in the project makes impossible to test all products. These techniques become also relevant when coping with software evolution. Software evolution, and thus, feature model evolution happens because new requirements are introduced in the development chain as the time goes by. In this scenario, granting the safeness of a product line is a must, before, during and after the evolution actually happens. In this case the rationale about having a focus with the evolution and testing is that after the keyworking process some papers touch in both sub-areas, probably, because testing is the main aspect to consider when evolving variability intensive systems as shown in Chapter §1.

From the set of reviewed papers, we identified more than sixty-two papers referring to SPL testing and evolution (see Figure §4.3). We also noticed that the works coping with testing costs reduction are focusing in scenarios where there is more than one objective –maybe contradictory– to be satisfied at the same time. This is, to optimize different aspects of the same test-suite. For example, to find the test-suite that minimizes the testing cost while maximize the market-share summation of the products within it.

Recently, researchers have been looking for the most convenient genetic algorithm to test and select best SPL products, finding that IBEA is the fastest algorithm [151] when coping with multi-objective testing objectives. Moreover, Heron et al [105], managed to do the same, but pruning at the same time. This is, to prioritize not products but T-wise covering test-suites.

In terms of evolution management different automated analysis have been proposed to grant the safe transition between the different evolution phases of the product line. In this area we can find several research opportunities. For example, a challenging task is to cope with implementation artifacts and variability knowledge to evolve the product line while supporting products from the previous software product line version. A motivating scenario for this can be found in the ubuntu open-source community. For example, there is no automated support when supporting users of a ubuntu distribution after

the distribution transition to a newer version by for example offering packages offering similar functionality but compatible with the newer OS version.

4.3.3 Product configuration and derivation

The configuration of feature models can be defined as the process of selecting and deselecting all features –those that difference one product from another– in a feature model. This is, to take all decisions needed to discriminate the desired product from the set of products encoded in the feature model. The next step in the process of composing a working product –derivation process –, is to compose and orchestrate the artifacts so the product provides the desired functionality and meet the quality requirements.

In the last five years, researchers explored new configuration mechanisms for building software product line products. For example, the use of feature-oriented techniques to implement software product lines have been extensively documented [9]. Also, researchers pushed forward the scalability of product derivation, making it more responsive [151].

Traditionally, configuration technologies are applied into a closed software product line context. This is, a context where all the decisions for configuring a product are taken by a single –or a very small group of– person which have access of all SPL variability documentation. An emerging challenge in this area is to configure a diversity of distributed software product line descriptions. As the product lines grow, configuring and maintaining them become an unfeasible task for a single person. Also, different stakeholders and privacy policies encourage the use of visibility restrictions of the configurable parts. We think that there is still work to be done in this area like, for example, enable the parallel configuration of those open and distributed product lines.

4.3.4 Variability and modeling expressiveness

Encoding the variabilities and commonalities of a product line requires to find a trade-off between the expressively of the language and its usability. Several proposals appeared to cover different domain specific requirements. For example, new relationships between features have to be considered when encoding Debian distributions[68] or cloud computing systems variability[73].

Also language constructs have been introduced in different languages to ease off the analysis of feature models and to extend the expressively of SPL. For example, FAMILIAr[4] introduced the forall construct to implement global

constrains over all feature model variables. Also, Clafer [10], introduced concepts coming from the class diagrams to represent even more complex scenarios.

Research opportunities are appearing as new domains are benefitting from AAFM. Also, because of the growing complexity of those models, more information is required to properly reason over such as big models. For example is still undetermined how to encode and manage the inter-model relationships between the different layers defined in CVL.

4.3.5 Multi-model variability analysis

The automated analysis of feature models started by only considering one feature model description at time. Again, the more the SPL become larger, the more complex reasoning techniques are needed to extract information from feature models. Yet CVL [44] introduced the disambiguation between SPLC features and the base model (a.k.a., feature realization artifacts). But also, because of the need of describe an variability intensive system in smaller artefacts so it can be maintained by different practitioners, new operations to merge models appeared, thus introducing new feature model operations such as the model merging or the slicing[5].

Different proposals coped with multi-model product lines. Moreover, having each model encoded in a different variability language. For example, Dhungana et al [57], proposed the distributed modeling of product lines providing examples coming from the automotive industry. New operations have been provided to, for example, determining the set of implementation artifacts that supports a concrete feature specification [115].

When coping with multi-model product line descriptions there is a lack of support for quality attributes. For example, there are no techniques to show how domain in quality attributes are reduced because of the selection of certain features at specification level. Also, the distributed analysis of feature models have not been yet explored. This is, slice first the model, then, distribute the model in the cloud, execute the analysis operation and finally merge the results. Therefore, being able to cope with models like the linux kernel one with more than 8000 features rapidly.

4.3.6 Reverse engineering of feature models

When developing a software product line, most companies do not start with the whole set of products developed and being maintained at the same time but only a few. This is, first, individual products are developed one after another and at certain point, when the number of similar products is big enough, the company transition to a SPL engineering approach. Therefore, when transitioning to a SPL-like approach, practitioners need to encode the variability in the best manner possible. For example, choosing between different possible topologies for a feature model. This process is known as reverse engineering feature models [3, 48, 78, 105, 161, 190].

The last 5 years have been fruitful in this area. Researchers have been able to extract the variability encoded in product comparison matrices[150]. Moreover, genetic algorithms have been used with good results extracting models from product descriptions[105].

First, as the variability is getting complex, attributed feature models are becoming more used, making the reverse engineering of attributed feature models a handy tool when applying AAFM techniques to far-from-spl feature models. Second, normally more than one variability model is used to describe the variability of a product line. Reverse engineering the relationships with more than one model is still an open challenge.

4.4 Research type

Figure §4.4 shows the distribution of papers classified by research type. In this distribution the most papers fits in the category of evaluation research. It is interesting to see that most research is not being validated.

4.5 Foras analysis

In order to answer the RQ2 and detect the forums where researchers working on automated analsyis of feature models are publishing we obtained the Table §4.1 from the set of publications. Table §4.1 shows the most used ones, above, you will find a large list list of conferences and workshops being used by area researchers. Note that in the Appendix §16 you will find the whole list of foras we found.

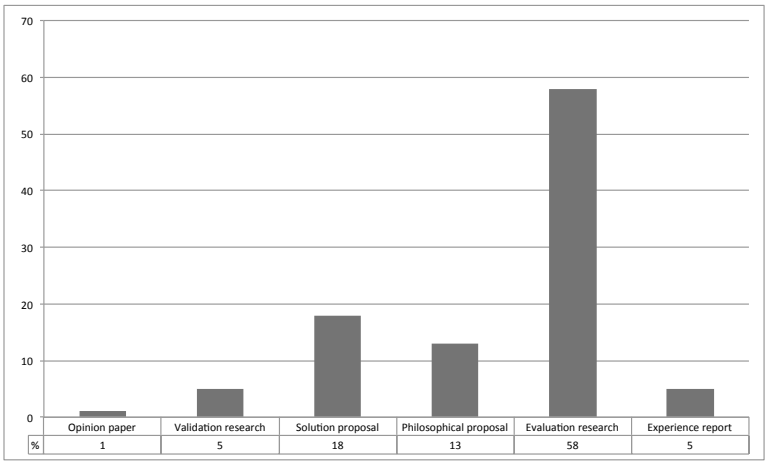


Figure 4.4: *Distribution in percentage of research type.*

International Conference on Software Reuse (ICSR)	4
International Workshop on Formal Methods and Analysis in Software Product Line Eng. (FMSPLE)	4
International Software Product Line Conference (SPLC)	22
International Conference on Model Driven Engineering Languages and Systems (MODELS)	5
European Conference on Software Architecture (ECSA)	2
International Workshop on Variability Modeling of Software-Intensive Systems (VAMOS)	25
International Conference on Automated Software Engineering (ASE)	12
International Conference on Advanced Information Systems Engineering (CAISE)	2
International Workshop on Product LinE Approaches in Software Engineering (PLEASE)	3
International Workshop on Feature-Oriented Software Development (FOSD)	3
International Conference on Generative Programming and Component Engineering (GPCE)	4
VARiability for You Workshop: Variability Modeling Made Useful for Everyone (VARY)	3
International Computer Software and Applications Conference (COMPSAC)	3
Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)	2
International Workshop on Model-Driven Approaches in Software Product Line Engineering (MAPLE)	8
International Conference on Fundamental Approaches to Software Engineering (FASE)	3
International Conference on Software Engineering (ICSE)	5
European Conference on Software Maintenance and Reengineering (CSMR)	4
International Conference on Search Based Software Engineering (SBSSE)	2
Workshop on Services, Clouds, and Alternative Design Strategies for Variant-Rich Software Systems (SCArVeS)	2
International Conference on Web Services (ICWS)	2
International Conference on Service-Oriented Computing (ICSOC)	3

Table 4.1: *Conferences & workshops with more than one publication.*

Finally, regarding the publication in journal we obtaining the following set of papers which are shown in Table §4.2 among the number of related papers published on it.

Software and Systems Modeling	3
Software Quality Journal	5
Science of Computer Programming	5
Information and Software Technology	6
Journal of Visual Languages and Computing	1
Journal of Object Technology	1
Computer	1
IEEE Transactions on Software Engineering	4
Journal of Systems and Software	4
International Journal of Digital Content Technology and its Applications	1
Information Systems	1
Journal of Theoretical and Applied Information Technology	1
Journal of Universal Computer Science	2
Expert Systems with Applications	3
International Journal of Emerging Trends & Technology in Computer Science	1
International Journal on Software Tools for Technology Transfer	3
International Journal of Software Engineering and Knowledge Engineering	1
IET Software	1
ACM Computing Surveys	1
ISRN Software Engineering	1
ACM SIGPLAN Notices	1
Studies in Computational Intelligence	1
International Journal of computers and technologies	1
International Journal of Information System Modeling and Design	1
Advances in Intelligent Systems and Computing	1
Electronic Notes in Theoretical Computer Science	1
IFIP Advances in Information and Communication Technology	1
Procedia Computer Science	1
Journal of Software Engineering	1
Science China Information Sciences	1
Frontiers in Artificial Intelligence and Applications	1

Table 4.2: *Journals and number of papers from the survey.*

4.6 Discussion

The surveyed research points out that the automated analysis of feature models is a research that is getting mature. However, the majority of the research is still published in workshops and conferences instead of journals. Concretely the percentage is 76 % in conferences and workshops and only 22.8 % in journals being the rest in books and other publications forms.

Regarding the distribution depending on the research papers the main fact detected is that feature model practitioners are not validating the research as much as they evaluate it. This actually should encourage the community to work more close to the industry and provide better ready-to-the-market solutions instead of techniques and prototypes. We suspect that this is intimately related with the fact that there are only a few industrial test cases and projects reported by the researchers.

When taking a look to the distribution of the research focus there are no much surprises as the areas with more work are, first "Configuration and feature oriented techniques" and second, "Testing and evolution". We suspect that this is because the areas that can provide more profit to companies. However, the new applications of feature modeling techniques now are being applied to far-from-spl areas, this points out that while the industrial applicability of SPL techniques into industry is not that strong, there are other areas such as cloud computing that can benefit from the work done in the past.

The distribution across the research foci is showing that most of proposals are focusing either in proposing methodologies or focusing in the modeling aspects of the SPL live cycle. Again, this may mean that the area is not yet enough mature to focus more in the last phases of the development, however, this is also contradictory given the number of papers focusing in evolution.

4.7 Summary

In this review we pinpointed the main trends in the area as well as shown a picture of the current state. Our aim is that this review will guide and help in the rest of chapters included in this thesis document. Also, we discovered new forums where publish our research and where to dig in when looking for related work.

We discovered that there is still work to get done in the area of AAFM. For example, more industrial applications beyond software product lines must be

shown and validated. Moreover, it would be interesting to see the possible benefits of applying AAFM for images recreation, cloud management, operating systems dependency problems detection and so on.

Chapter 5

Research gaps

Twenty years from now you will be more disappointed by the things that you didn't do than by the ones you did do, so throw off the bowlines, sail away from safe harbor, catch the trade winds in your sails. Explore, Dream, Discover.

Mark Twain, Writer and humorist

In the previous chapter we performed a mapping study that update the state of the art. Moreover we hightlited the current main trends in feature model automated analisys. In this chapter, we go through the main research problems existing in each trend analyzing the current solutions and pointing out possible research oportunities.

5.1 Introduction

The analysis of feature model is a research area that have been gaining hype in the last 25 years. Moreover, different researchers already started to apply automated analysis of feature model in different contexts such as cloud systems [73]. In Chapter §4 we realized that there were still research gaps in the different trends and sub-areas of the automated analysis. In this section we explain the current research approaches and its weak points.

Variability intensive systems introduces a diversity of new challenges when using automated analysis techniques. In this thesis we focus on the following ones : i) variability intensive systems encode non-boolean variability information, for example, video-analysis systems require to encode physics related information as floats; ii) variability intensive systems require to use roles and permissions to support segregation of duties when more than one practitioner configure the same ecosystem and iii) variability intensive systems evolve and change over time in multiple ways but there are some that would provide more benefits than others.

The objective of this chapter is to motivate the need for new methods and mechanisms that while enhancing the automated analysis of feature model are applicable to variability intensive systems. We focused in three main facets of the software. Namely, evolution, testing and configuration. In the next section we depicts the challenges we found in these facets of software while dealing with variability intensive systems.

5.2 Problems

The main contributions of this thesis were motivated by the following problems:

Evolution of variability intensive systems. Evolutions affects software by introducing new requisites in the development chain as the time goes by. Variability intensive systems are not an exception, moreover, because of the large amount of products they encode there are configurations from previous versions than aim to transition to new model version. Actually, different configuration path can achieve this transition, being some more expensive than others. In this context we focused on the problem of identifying the path that satisfy a constraint while minimizing the cost to spend in the transition.

Testing variability intensive systems. The large amount of products en-

coded in a variability intensive systems makes impossible to test them all. In this situation, is appealing to reduce and prioritize the tests we are going to execute. Also, there is information that can be linked to the features, thus, exploited and used to improve the testing process such as market-share metrics or relative cost impacts. The problems we faced refering to this information are, first, how to encode such new variability flavors with non-boolean information in a manner that can be automatically analyzed. Second, to extend and apply automated mechanisms over the encoded information retrieving only useful testing information.

Configuring variability intensive systems. Configuring variability intensive systems in s daunting task for several reasons. First, the large number of products makes impossible for one person to take all configuration decisions in a rapid manner. Second, each practitioner define the variability with the variability models they are more confortable or used to. Two options can be considered, either to implement an standard [44] or to allow the configuration orchestration of multiple and diverse variability models. In this thesis we pushed forward the second strategy to allow collaboration in between practitioners.

5.3 Current Solutions

In this Section we will present the strong and weak points we found in each of the research areas detected in the previous Chapter.

5.3.1 Evolution of variability intensive systems

Model-driven feature model evolution. A number of approaches have looked at the development of modeling tools to support feature model evolution. Pleuss et al. [31, 133] model coherent sets of changes to a feature model as model fragments and allow modelers to describe evolved versions of feature models at future points in time. Further, the underlying model-driven tooling allows developers to check the correctness of the evolved models or interactively evolve the model. Whereas these existing approaches focus on the user-interface modeling and constraint-checking aspects, there is a lack of automated mechanisms for optimizing the planning steps of future evolutions of configurations. For example, Pleuss et al.'s techniques do not provide configuration evolution optimization capabilities or automated non-interactive evolution based on objective functions.

Re-configuration approaches. Other research has looked at re-configuration or SPL error correction in a number of different scenarios. For example, Xiong et al. [187] researched eCos configuration problems, proposing a set of solutions for a set of concrete configuration errors. These works miss to consider multiple-steps configration. This is configurations that span over several time steps.

Managing changes using dynamic SPLs. Rosnmuller et al.[147] investigate unforeseen changes due to the run-time swapping of components in a SPL. This works focuses on evolution of variants within the same SPL. Feature model drift allows modifications to the structure of the SPL which is not covered in the research by Rosenmuller et al.

Supply-chain product-lines. Hartmann et al. [77] investigate methods of building models that incorporate the variability and constraints of multiple suppliers into a product-line feature model. Hartmann's work focuses on the modeling aspects related to capturing and maintaining the constraints from multiple suppliers whereas no mechanism to reason about the constraints over time are provided.

Understanding configuration over time. Elsner et al. [62] have looked at variability over spans of time and the issues related to understanding when and how variability points relate to each other. In required to focus on automating three key tasks that Elsner et al. identify as needed for managing variability over time. Specifically, providing capabilities for automating and optimizing tasks that Elsner et al. term: 1) proactive planning, 2) tracking, and 3) analysis. Whereas Elsner et al. focus on general identification of the issues in managing variability over time, it misses to focus on providing a framework for automating the specific tasks.

Quality attribute evaluation. Feature models can be attributed with non-functional information such as price, cost, and time to deploy. Johansen et al.[87] proposed to use different weights depending on the importance of each software product line, in that way, they give more importance to some features than to others. No solutions enable maximization of different model properties such as the number of different features involved in a configuration, or maximizing concrete quality attributes. Several techniques have been proposed for evaluating quality attributes [65, 85, 124] to guide a configuration process. These techniques provide a framework for assessing the impact of each feature selection on the overall capabilities of the configured system. As a result, quality characteristics, such as reliability, can be taken into account when selecting features. These techniques are also designed for single step

configuration processes.

Step-wise refinement. Batory[11] describes AHEAD, a technique for the configuration of of SPLs. AHEAD utilizes step-wise refinement, in which SPLs are configured iteratively. This miss to select additional features over the course of multiple-steps in order to reach a target configuration.

Feature edits. In the work presented by Thum et al. [174], the authors present a catalog of possible changes that can be applied to a feature model in the process of model evolution. The authors also discuss approaches for determining if the new model is a subset or a super-set of the original feature model. This work differs from ours in many aspects. A weak point of this paper is to not consider multi-step changes nor identify paths between configurations across multiple steps.

5.3.2 Testing of variability intensive systems

Software product-line testing In order to reduce the cost of testing a software product line, researchers have presented several approaches for reducing the number of tests to execute, and therefore the cost of the entire testing phase. For example, [43] presented an overview of possible strategies for reducing testing costs. [58] present a literature review of strategies for test software product lines. These works discuss the two principal mechanisms when testing software product lines–domain artifacts testing and application artifacts testing. However, they do not evaluate artifacts testing, where combinatorial optimization has proved to be one of the most promising approaches.

Software product lines are an approach for building a configurable software platform that can be adapted to varying requirement sets. A key component of a software product-line is that there are a set of common components, as well as points of variability that can be adapted to a given requirement set. Some research [24, 93, 116] has investigated testing techniques that can be applied to software product lines. These software product line testing techniques provide varying algorithmic approaches to ensuring that the entire configuration space of the software product line is covered. For example, [93] presented a method to reduce the efforts of executing a test over the whole set of products encoded in a product-line. This is done by analyzing the model and only execute the test one time in the parts not affected by variability. Is still unknown how to to derive a set of the underlying software product line configurations to test on top of.

Pair-wise testing. One of the most appealing approaches has been pre-

sented in [41]. Cohen et al. first explain the importance of creating good covering arrays for the set of products encoded in a product-line, and later in [42] presented specific algorithms to generate the covering arrays and provide empirical results. Several others have utilized this general method[42, 86, 98, 130, 131]. For example in [125], authors prosed the use of CSP for generating pair-wise feature permutations. Our proposal is more ambitious than the one presented by Oster et al., by enabling not only the use of t-wise pruning but also more complex functions such as the cost of executing the tests in the cloud. These approaches can be complementary. For example, by using a pair-wise function as the cost function, we can discard some t-wise pairs with some reasoning over the attributes of the features involved in the product. Moreover, [87] focuses on value guided testing by generating covering arrays. In that approach, sub-product-lines have different weights, thus some products are more critical than others when testing.

Cost guided testing. The cost of software development is one of the main concerns when developing software. There are several proposals that present solutions to address the problem of testing costs [26–28, 42, 125, 130, 131, 148, 165, 167, 185]. There is a lack of proposals to minimize the testing cost while maximizing the value of our tests.

Configuration management & analysis. She and Lotufo et al. have investigated techniques for modeling the features and configuration rules embedded in the Linux kernel[106, 160]. In their work, they use similar feature modeling approaches to manage the variability in the Linux kernel. We focus on how to model the variability of configurable platforms and derive valid configurations. However, a key difference with our approach is that it is specifically focused on combining this configuration data with a market-share model in order to derive platform configurations for testing.

Multi-objective optimization problems. There are circumstances where it is important to balance trade-offs between different testing objectives. These problems are known as "multi-objective" problems. Most solutions for these problems are based on evolutionary algorithms [39, 51].

Recently there is a trend of applying these techniques to software product line testing. Because of their similarity with our research, we have developed a deeper comparison between these approaches. The results obtained by this comparison are shown in Table §5.1. Concretely, we compare; i) if the approach uses multi-objective solving techniques. This is, if multiple functions, maybe conflicting, are used for the same testing operation; ii) the testing operation supported (pruning, prioritization and packaging); iii) if the solution allows user defined functions; iv) the support for quality attributes; v) the completeness of the solving techniques, this is if the approach explores the

whole solution space or uses heuristics mechanisms, and vi) if the implementation is available and if supports complex cross-tree constraints (constraints over attributes) and attributes with ranges. Note that the techniques described here are not focused on product prioritization, pruning, or packaging but techniques that can be applied to other multi-objective problems such as the configuration of a product-line.

[152] explored the existing evolutionary algorithms to find Pareto optimal solutions. The research found that IBEA algorithm provides the best performance when dealing with feature models. This work presented a comparison between different multi-objective solving techniques based on evolutionary algorithms and enables the prioritization of objective functions over features. The main difference of this approach compared to our research is that instead of heuristic based methods, such as evolutionary algorithms, our solution uses a CSP and support attributes.

A different approach was presented by [81]. In this work, authors proposed the configuration or generation of test-suites by using objectives over features. An interesting point of this approach is the generation of test-suites based on the pair-wise coverage they offer. Again, the main difference with our solution is the approach used (Evolutionary or CSP) and the attributes support.

Finally, [123] used an exact solver to obtain the set of products based on attributes values. The main differences between these approaches and our approach is the use of different solving techniques that explore the whole solution space. Another notable difference is that our research needs to allow for constraints that relate attributes to features. Note that none of the approaches explicitly support prioritization and not are focused on testing.

Knapsack algorithms. Knapsack problems and bin packing problems have been studied for decades [40, 110]. In our research, an exhaustive solution based on dynamic programming has been used to select the set of hardware configurations that maximize market-share without exceeding a maximum budget when testing. Researchers have proposed other methods to solve the knapsack problem based on heuristics [6]. These methods provide a good approximation of the maximum value of the knapsack, but have much better algorithmic time complexity.

Value-driven development. Companies aim to maximize profit when developing software. Value-Driven development is a series of processes that companies can follow to focus development and testing to maximize profitability. There is previous research on this topic [27, 28, 167]. [165] proposed

Approach	[152]	[81]	[123]	Research gap
Multi-objective solver	●	●	●	○
Pruning support	○	⊘	○	●
Prioritization support	○	○	○	●
Packaging support	○	○	○	●
User-defined testing functions	○	○	●	●
Attributes support	○	○	●	●
Completeness	○	○	●	●
Implementation available	○	●	○	●
Complex constraints	○	○	○	●
Attributes with Ranges	○	○	○	●

● addressed as goal, ⊘ addressed but with restrictions, ○ not regarded as goal

Table 5.1: *Multi-objective related work comparison.*

the use of a Return of investment (ROI) metric in order to guide the different stages of software development. The ROI metric provides a mechanism to quantify the profitability of developing a software product at various stages in the software development lifecycle. To maximize testing ROI, [167] proposed different ways to prioritize test-case execution. In this research, the market-share metric has been proposed to maximize the number of users reached by the platform configurations that a mobile application is tested on. This approach is a specific type of ROI analysis for value-driven development and complementary to prior research.

5.3.3 Configuration of variability intensive systems

Staged configuration. Czarnecki et al. [47] describe a method for using staged feature selection to achieve a final target configuration. Their multi-stage selection considers cases in which the selection of features in a previous stage impacts the validity of later stage feature selections.

Configuring multi-step problems is complementary to Czarnecki et al.'s work since it (1) examines the production of a feature model configuration over multiple configuration steps and (2) provides a general formal framework that can be used to perform automated reasoning on staged configura-

tion processes. There is a lack of technique to reason about other multi-step configuration processes that do not fit into the staged configuration model such as the avionics example used in chapter §8.1. Hwan et al. [83] have looked at mechanisms for synchronizing specializations of feature models as changes occur over time. This problem is similar to the one we target in this thesis, but we focus in a different and complementary aspect of the problem, which is reasoning in the face of changes to the feature model over time. Both synchronization and automated reasoning in the face of changes to the underlying feature model are needed and each approach addresses a different aspect of the problem.

Classen et al. [36] have investigated creating a formal semantics for staged configuration. Moreover, they provide a definition of a configuration path through a series of stages for a feature model. However, existing solutions can not produce a complete configuration at multiple points in the configuration process.

Automated single-step configuration. Several single-step feature model configuration and validation techniques have been proposed [12, 20, 25, 33, 108, 182]. These techniques use CSPs and propositional logic to derive feature model configurations in a single stage as well as assure their validity. These techniques help address the high complexity of finding a valid feature selection for a feature model that meets a set of intricate constraints.

While these techniques are useful for the derivation and validation of configurations in a single step, they do not consider feature configuration over the course of multiple steps. In many production scenarios (such as the avionics example from Chapter §8.1) the ability to reason about configuration over multiple steps is critical.

Bosch et al [30] describe several evolution patterns that appear in the configuration of SPLs. These patterns incorporate the effects of manipulating variation points in regards to time and resource consumption. In contrast, no solution is handling this evolution variability by spreading feature selections over multiple stages, such that an introduction of a new variation point can be taken into account.

Configuration workflows. Hubaux et al.[82] presented a formalism to determine the work-flow required to configure a feature model in multiple steps. However, Hubaux et al's work does not investigate feature model drifts or automated derivation of a configuration path between a starting and final configuration nor multiple practitioners configuring at the same time.

5.4 Discussion

From the analysis of the previous work we detected that the testing, configuration and evolution of variability intensive systems have not been addressed when having to cope with non-boolean information and complex constrains. Also, this points out that existing mechanisms were not enough to reason over quality information such as floats when testing. Concretely we identified the following concrete research gaps to enable the evolution, testing and configuration of variability intensive systems:

Testing of variability intensive systems. From the set of reviewed solutions, we noticed that there were two main activities requiring research efforts. Researchers were pruning and prioritizing the products to test only taking into account certain feature model characteristics such as the feature commonality. Second researchers were narrowing the products to test by using t-wise methods, thus, forgetting about the non-functional properties of the variability intensive systems that can be used to better scope the test-suite. In this dissertation we tackled this problem by proposing the use of nonfunctional information such as cost, value and other per-feature non-boolean information. This is, to include business information as part of the testing.

Configuration of variability intensive systems. While there have been an extensive work in the configuration of individual models, we did not find papers coping with the configuration of diverse and distributed variability models. Techniques such as the the staged feature models can certainly help in this task. To enable the configuration of variability intensive systems we developed a solution that can configure a variability intensive system relying in the current single-and-not-distributed analysis tools such as FaMa, FaMaOVM or Dopler.

Evolution of variability intensive systems. Evolving variability intensive systems require mechanisms and tools that optimize the configuration path in between different configuration. This is, to optimize the transition from configurations existing in previous version to configurations in the current model version.

5.5 Summary

In this section we motivated the needing of our research. We went trough the related and previous work related to our research highlighting the main research gaps to address in this thesis work. In the next chapters the reader

can find the concrete solutions we implemented to address those problems.

Part IV

Contributions

Chapter 6

Testing: pruning, prioritizing and packaging

El caudal de la labranza siempre rico de esperanza.

Dicho popular, Andalusian people

Software product-lines are used to develop a set of software products that, while being different, share a common set of features. Feature models are used as a compact representation of all the products (e.g., possible configurations) of the product-line. The number of products that a feature model encodes may grow exponentially with the number of features. This increases the cost of testing the products within a product-line. Some proposals deal with this problem by reducing the testing space using different techniques. However, a daunting challenge is to explore how the cost and value of test cases can be modelled and optimized in order to have lower cost testing processes. In this chapter, we present TESting vAriAbiLity Intensive Systems (TESALIA), an approach that uses automated analysis of feature models to optimize the testing of variability-intensive systems. We model test value and cost as feature attributes and then we use a constraint satisfaction solver to prune, prioritize and package product-line tests complementing prior work in the software product line testing literature. [†1]

[†1] This chapter is based in [71] and part of this material is in press in the SQJ journal

6.1 Introduction

To represent the common and varying features within a product-line, a variety of variability modeling techniques have been proposed [163]. However, feature models are one of the most widely used techniques to model the variability of a software product-line [92]. To encode the set of all software product line products in a feature model's tree-like structure, a variety of parent-child relationships are used to represent the constraints governing the possible feature permutations of valid products in the product-line. The number of products encoded in a feature model grows with the number of features. Given n features and no constraints on valid feature combinations, there are 2^n possible products. To deal with this complexity, automated mechanisms are used to extract information from feature models, such as features present in every product. These automated mechanisms are known as *automated feature model analysis* techniques [19].

Some software product-lines, such as those describing operating systems [68, 160], have thousands of features [13]. When a product-line has an exponential number of products relative to its feature count, testing all or a large percentage of its products is an error-prone and expensive task, both in terms of time and computing resources. To address the issue of not being able to test all the possible products of a software product line, combinatorial testing [121] can be used. Combinatorial testing reduces the number of tests to execute by taking advantage of the shared features across different software product line products. Roughly speaking, the hypothesis behind combinatorial testing is that if a given feature has already been tested, then the testing process can skip further tests containing this feature. A variety of heuristics can be used to decide which feature combinations to test, however, researchers have primarily used t-wise heuristics for pruning the set of products to test.

The testing phase in a software engineering process can be time-consuming and expensive [15]. One of the parameters to be considered when selecting test cases, is test case cost and effort. The same is true in software product-line engineering. Past literature has investigated techniques for reducing testing effort in software product lines [26–28, 81, 86, 87, 98, 123, 125, 130, 131, 148, 152, 165, 167, 185]. Traditionally, testing cost has not been modeled explicitly. For example, when applying t–wise based approaches [79, 80, 87, 98, 125, 130, 131], the cost to test a feature is extracted from the number of product configurations that include a specific feature. Modeling cost information in a more explicit way can help to reduce the number of products to be tested in a software product line. For example, if we are testing in the cloud, the time and budget required for executing a test may vary between different executions. Time and budget are examples of testing cost. Current approaches for testing software

product lines do not consider these situations where test cost is not uniform.

When the time to test all products in a product-line outstrips the available testing budget, determining how much value is obtained by running a test enables the prioritization of the products to be tested. When there is limited time for testing due to deadlines, test prioritization enables the critical parts of the software to be tested when not all tests can be executed [166]. Test value [27] can be measured in different ways such as the measuring the feature's relative importance in terms of the project requirements (e.g., points on associated user stories) or deriving importance based on the time left until its expected completion date in the project schedule. The value of a software product changes as time goes by. For example, a new mobile phone may be widely used when it is first introduced in the market. Over time, as its feature set becomes older and less attractive, it may have fewer and fewer users. Therefore, the value of thoroughly testing an application running on a particular phone may provide more or less value depending on when the tests are run. Mechanisms to model the test value in software product lines can be used to properly reduce the amount of products to test while maximizing the value of the tests that are run.

To improve the testing process researchers have primarily proposed two mechanisms; i) *test pruning*, to selectively execute tests based on test-cost when there are a large number of tests and [121] ii)*test prioritization*, to ensure that the most important tests always run first in case testing is interrupted [166]. With the first approach, developers can reduce the number of tests to execute while maintaining good test coverage. With the second approach, developers execute the most valuable tests first in order to ensure that critical software components are tested. Moreover, when companies use test-driven development approaches [14], the large number of test executions when using software product lines can result in high costs. Being able to fix a maximum budget for a test-suite enables developers to bound the cost of the testing phase while maximizing testbed value.

Automated analysis of feature models is the process of extracting information from feature models using computer aided mechanisms [19], such as constraint solvers. More than thirty different approaches have been proposed to extract varying kinds of information from feature models, such as how commonly a feature is used or the number of unused features [19].

In this chapter we propose to use automated analysis of feature models to automate the pruning and prioritization of the tests to be executed within a product-line. We present TESALIA, a method to analyze the set of products encoded in a software product line, as well as feature value and cost information, in order to prioritize the products to be tested. We also propose a method

to build a product test suite that adheres to a fixed testing budget while maximizing the value of the tested products. This chapter sumarizes the following contributions:

- A mechanism to store, update and reason about value and product cost data.
- A CSP-based approach to prune the product set encoded in a feature model. This approach extends prior work [79, 80, 87, 98, 125, 130, 131] by enabling the use of a variety of objective functions for pruning test cases. For example, a 2-wise function or a function based on test execution time can be used to prune the tests.
- A CSP technique to prioritize the list of products to be executed in order to maximize testing value.
- A technique for packaging tests with cost and value data to bound testing cost while maximizing test suite value.

6.2 Motivating scenario

Figure §6.1 shows the motivating scenario that promoted this research. Mobile phone applications *("apps")* are typically executed on a variety of different underlying mobile platform configurations, such as different versions or configurations of Android or iOS. Each platform configuration has different mobile platform features, such as screen resolution or communication capabilities (*e.g.*, 3G, LTE, etc.) that the software must interact with and may be configured differently depending on the platform version. For example, iOS 6 on the iPhone 3GS does not have turn by turn directions but iOS 6 on the iPhone 5 does. Also, the iPhone 3GS-4S has the same screen aspect ratio but the iPhone 5 does not. The Android emulator [†2], which allows developers to emulate the configuration options of real-world devices, currently supports 46 different mobile platform features, which (presuming all user configurable features could be combined with no restrictions) potentially leads to 2^{46} different platform variant configurations. The high variability that exists in the Android ecosystem makes testing hard. Being able to describe the variability of the Android ecosystem as a software product line would allow developers to apply existing and new testing techniques to optimize testing strategies.

[†2] http://developer.android.com/tools/help/emulator.html

Because of the large number of in-use platform configurations, it is difficult for developers, regardless of development team size or proficiency, to test their software products on all or even most platform configurations before release. One of the most common approaches is to pick a set of the most popular mobile devices and then to test exclusively for that subset of devices. For example, the Android Skype application is installed on thousands of unique Android platform configurations [†3], but the Skype application is only officially certified to work on fewer than 25 Android devices[†4].

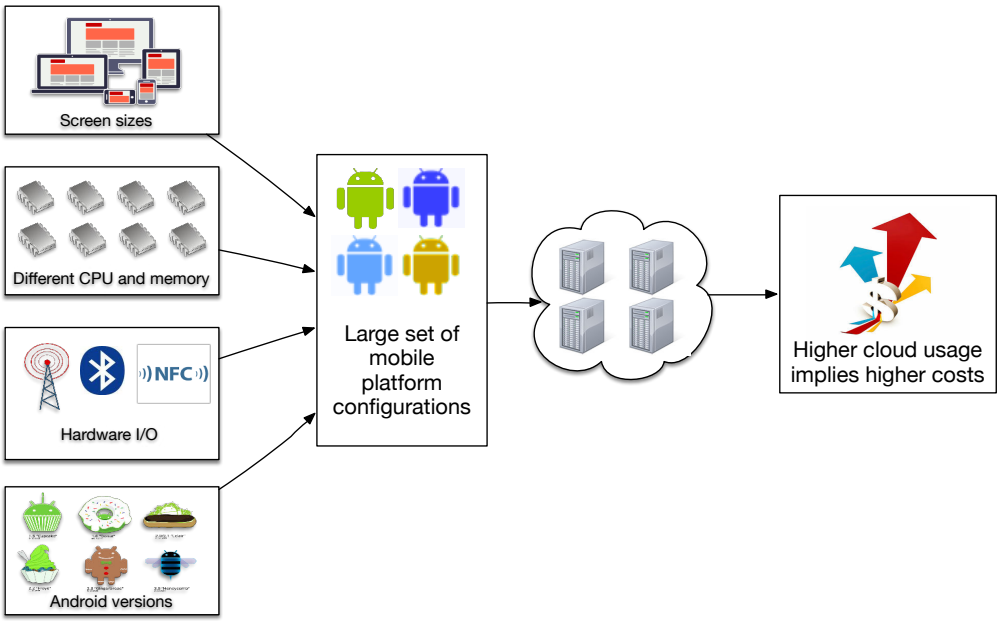


Figure 6.1: *Android variability impacting testing costs.*

A primary challenge of testing on a large number of mobile platforms is the cost of acquiring so many unique smartphone devices. As an alternative to testing on actual devices, developers frequently turn to platform emulation to mimic real-world platform configurations. However, running a complete suite of tests on hundreds of uniquely configured platform emulator instances requires substantially more computing power than the average developer possesses. An option to address this problem is to use the cloud as a supporting platform for running virtual instances of the emulating configurations.

To guarantee the correct operation of Android applications developers can

[†3] http://opensignal.com/reports/fragmentation.php
[†4] http://www.skype.com/intl/en-us/get-skype/on-your-mobile/skype-mobile/android/

adopt a test-driven development process [14]. Test-driven development relies on the repetition of a set of tests in short development cycles (e.g. every time we release a new version, every time we add a new functionality). When developing applications that potentially will run in a wide range of different platform configurations users need to select which configurations will be used to run the tests. Usually developers may want to improve the value of their tests by defining different metrics such as market-share impact, which is the number of real devices (mobile platform features) and mobile platform features covered by the tests. Market-share analysis can be used to help determine which tests to run and automate this testing in the cloud.

Goal: Minimize the probability of encountering an untested feature configuration when installed on a device in the wild.

Ideally, a producer of a mobile application would like to ensure that, when their app is installed on an arbitrary mobile device in the wild, the app has been tested with as many of the device's platform feature configuration options (*e.g.*, screen size, API version, etc.) as possible. The more untested feature configurations of the device, the more likely there is a chance that an unknown bug may emerge and that the app will not function correctly. A key question, therefore, is how to intelligently choose platform feature configurations to test on in order to minimize the chance of encountering a platform feature configuration in the wild that present errors. Moreover, since budgets for testing time and cost are limited, how do developers select the best mix of platform configurations to test on given their time and cost constraints?

Testing every possible platform configuration is not feasible due to the large number of platform feature configuration permutations. For example to test an application with similar requirements to the Facebook app would require testing every platform configuration that, at a minimum, includes varying parameters for the camera, camera APIs, GPS, and the nine different screen resolutions supported by Android at the time of writing this thesis. Those options represent 2^{11} platform configuration permutations to run the tests on, which may be expensive and time consuming.

The typical approach taken to handle this problem is simply to pick and test on the platform configurations of the most popular mobile devices. Picking popular mobile devices, however, may not be an effective strategy if the goal is to reach as many users possible with our app. For example, a popular new phone may have a higher resolution screen, that is not representative of the screen resolutions of the existing older mobile devices in use. Section §10.2 shows specific examples of where testing on the five most popular Amazon devices covers over 40% fewer platform feature configuration options of the in-use devices versus a more intelligently selected set of five platform config-

urations.

To reduce the complexity of testing software product lines, a number of challenges need to be addressed:

Modeling the variability in the mobile platform ecosystem. A daunting challenge of developing mobile applications is the rapid pace that the underlying mobile platforms and device hardware capabilities are evolving. This rapid pace of evolution has led to platform fragmentation, which is the large variability in platform configurations used on different mobile devices in the market. Fragmentation may be caused by differences in manufacturer modifications to an open-source platform, such as on Android, or the evolution of a single manufacturer's product-line, such as Apple's iOS products. For example the bluetooth API changed substantially between Android versions, making apps designed for initial versions fail in current flag-ship devices. Many of these configuration variations, such as differences in how the API to access a user's contacts functions, can directly impact the functionality of apps. The fragmentation creates issues when testing because of the large amount of testing required to ensure that the app works properly across the entire platform configuration space.

Defining the cost of testing a mobile application. Regardless of whether real mobile devices or emulator instances executed in a cloud, such as Amazon EC2, are used, developers typically do not have the time or money to test their app on top of every possible mobile platform configuration. A key issue, therefore, is determining how many and which mobile platform configurations to test a mobile app on without exceeding development time or cost budgets. For example, although a cloud can be used to test thousands of emulated mobile device instances, most cloud-computing providers charge fees based on the amount of CPU time consumed by an application, which can lead to substantial cost. For example, using Amazon EC2 [†5] to test an Android application that runs on top of 1000 unique platform configurations may require 1000 cloud instances. If these instances have, on average, a cost of $1.006 per hour and each test consumes $\frac{1}{2}$ hour, the total testing cost will be $503, or $1006 per hour. Thus, although developers *can* test their software on thousands of platform configurations, they must determine the number of configurations to test given their desired cost, time, and coverage goals (tests profits).

The cost of executing a test can be measured in different ways depending on a company's requirements. For example, it can be measured in terms

[†5] http://aws.amazon.com/s3/

of money invested for running an emulator in the cloud, or in terms of carbon emissions. Moreover, it also can be measured by executing a test and capturing the time required to run it or the number of lines of code executed.

Determining the revenue of executing a test. A key approach that developers often use is to leverage sales data from a vendor, such as Amazon, in an attempt to maximize the market-share of the devices that their app has been tested on. The overall goal of developers is to select a set of top selling phones from a period of time in an attempt to minimize the number of mobile platform configurations that the app is installed on that it has not been tested on top of. However, to date, these approaches to selecting which mobile platform configurations to test on are manual, ad-hoc processes that are not optimized. Further, these manual processes to selecting mobile platform configurations to test on top of do not consider the complex tradeoffs between development budgets, time, and market-share coverage. Finally, these existing approaches of selecting top selling devices to test on have not been evaluated for market-share coverage or compared to other approaches.

Different value functions can be used to achieve different testing scopes. For example, if we use the market share metric, our test will reach as many users possible but might leave out android installations without networking capabilities (e.g., no Wifi or cellular modem hardware). If we want to look for errors on rare devices we can try to maximize the number of mobile platform features covered by our tests.

Skype [†6] only certify their Android application for the most sold phones, that makes that when a new device appears into the market some hardware pieces may not have been tested with the application (e.g. in Sep'13 they have not tested the Nexus7 Google tablet). Therefore, this work proposes to maximize market share while meeting the cost boundary.

Prior work proposes addressing this tradeoff between testing cost and number of tested platforms by reducing the number of tests [94, 117] or automating the testing process [61]. This existing body of work has not extensively studied the problems unique to optimizing mobile software testing cost and the market-share coverage of the mobile platform configurations used for testing but it did in the SPL testing area. The number of different mobile platform features required to verify that our mobile application is error free is huge. To execute those tests in a reasonable amount of time the use of cloud computing is a must. However, the execution of those experiments might be

[†6] http://developer.android.com/tools/help/monkey.html

expensive making it unaffordable to mid–size companies. Know how much market–share our tests are covering is required by developers to know how may potential mobile platform features are still on the wild. Therefore, mechanisms and tools are being needed to optimize the number of required test that maximizes the hardware platform.

Later, we describe how we use TESALIA to model Android variability, add cost and value information, and prioritize, prune, and package tests. For the sake of simplicity, in the next Section, we use a simplified version of the motivating example to present the key ideas. This example is shown in Figure §6.2. The simplified example is based on a feature model with three concrete features (Audio, Wifi and Bluetooth), all the features have a "cost" attribute representing the relative cost within the product and a "value" attribute for specifying the relative benefit of having the associated feature in a product. To simplify, the attributes are expressed as integer values.

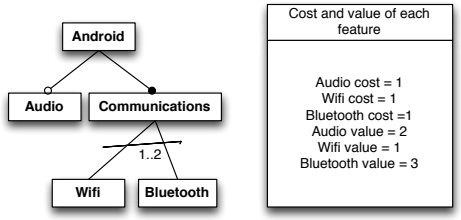


Figure 6.2: *Feature model example based on the motivating scenario.*

6.3 The TESALIA solution

The TESALIA (TESting vAriAbiLity Intensive Systems) approach provides a mechanism for finding the most valuable set of products to test while meeting a fixed budget (cost). First, the variability present in an software product line is described using a feature model. Second, the feature model is attributed with information modeling test cost and value. Note that only features impacting test cost and value have to be attributed. Finally, using automated mechanisms, a list of products defining the selected product for testing is generated. In the remainder of this section, we present the key components of the TESALIA approach.

Figure §6.3 shows the TESALIA solution approach. The key benefits of TESALIA are that it enables i) the pruning of the possible products to test; ii)

the prioritization of the products to test based on value; and iii) the packaging of products into testable sets that meet a bounded cost constraint while maximizing the overall test value of the set. Each component of the solution can be executed sequentially. In other words, testing practitioners can decide either to execute all operations or only some. For pruning and packaging, developers must provide both a cost and value estimation function, based on feature attributes.

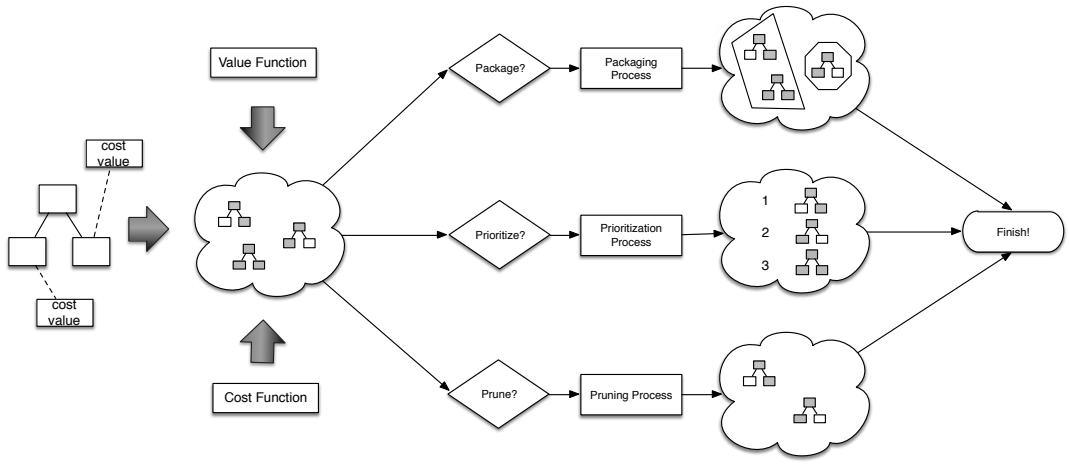


Figure 6.3: *TESALIA solution overview.*

6.3.1 Capturing testing variability with feature models

In order to optimize the number of tests to run in a product-line, computer-aided mechanisms are needed to describe the valid products that can be targeted at when testing. The number of different possible valid products in a product-line can grow up to 2^n, where n is the number of features. In order to describe the valid and non-valid products in a product-line, researchers use a variety of variability modeling approaches.

Although feature models have been used for modeling software product line variability, they can also be used to model other kinds of variability, such as cloud configurations [60, 73] or Linux–based distributions [68, 160]. Also, there is a recent trend to use feature models to describe the variability in an application's testing space [26–28, 81, 86, 87, 98, 123, 125, 130, 131, 148, 152, 165, 167, 185].

For example, as motivated in Section §6.2, when testing Android applications on multiple Android devices, there are a large number of potential feature permutations of the Android emulator, such as audio and connectivity capabilities, screen size or CPU type. With such a large number of permutations, it is easy to create an invalid emulator configuration for testing, such as selecting Android version 1.5 along with a screen resolution of WVGA, which is not supported by that platform version. Therefore, when attempting to test across a wide range of configurations, a method of defining the valid configuration options and automatically discarding invalid configurations is required.

TESALIA uses feature models to describe the set of feature permutations (testing variability) available for testing as some other have proposed [26–28, 81, 86, 87, 98, 123, 125, 130, 131, 148, 152, 165, 167, 185]. Having the testing variability described in a feature model makes it possible to use automated techniques to identify valid feature permutations [19]. Moreover, it allows automated reasoning about the evolution of the product-line over time [179], such as the cost of transitioning to a different configuration containing a new Android version. For example, the valid products described by the feature model presented in Figure §6.2 are shown in Table §6.1. The processes for constructing these feature models is beyond the scope of this thesis. However, there are a variety of techniques that have been described in prior work that can be used to reverse engineer feature models from product descriptions, such as domain engineering approaches [92] or computer-aided algorithms [105, 162].

Name	Android	Communication	Audio	Wifi	Bluetooth
P1	✓	✓	✓	✓	
P2	✓	✓	✓		✓
P3	✓	✓	✓	✓	✓
P4	✓	✓		✓	
P5	✓	✓			✓
P6	✓	✓		✓	✓

Table 6.1: *Set of products described by the example presented in Figure §6.2.*

6.3.2 Attributing feature models with testing data

Once the testing variability is captured in a feature model there are a number of prior approaches that can be used to reduce the number of products to be tested by applying, for instance, t-wise methods [79, 80, 87, 98, 125, 130, 131]. TESALIA uses feature attributes in the feature model to specify value and cost of tests to aid in pruning the testing space and later prioritizing the set of test cases to run.

Combinatorial explosion is a well known problem in software testing [76]. Because of the large number of software configurations, execution paths, and inputs, it is generally impossible to execute sufficient tests to completely test every execution path and input configuration of a software artifact. This problem also occurs in software product line testing because the number of software product line products is an exponential function of the number of features in the product-line. One of the goals of software product line testing is to select the best set of test cases that maximizes the chance of detecting a bug while reducing the testing effort. To do so, the *cost* and *value* of a test case must be measured to quantify its importance. In this chapter, we refer to the cost of a test case as a measure that quantifies the effort of executing a test case. This can be measured in terms of resources, such as budget, person-months, cloud computing time consumption, or other measures [15]. Additionally, the value of a test case is the measure that determines the importance of a test case. It can be measured in terms of number of tested features or market coverage [27, 189].

To add cost and value information to the set of products in TESALIA there are two possible approaches. The first approach is to specify the cost/value information for each product by executing the tests associated with the product and storing their cost or value for further use. The second approach is to manually attribute the model with the relative cost or value of each feature. However, it requires manually updating the information associated with each test and product, as well as updating this information as the software product-line evolves. The cost and value of each product can be obtained by using a function that provides the cost of each product by using the values of the attributes present in the model. This can be done by, for example, extracting usage information from the Android play store in the case of testing mobile apps. Benefits and drawbacks of each approach should be determined by domain experts in each case. However, there are studies modelling product costs of cloud services (measured in monetary units) [73] or modelling space restrictions when designing circuit boards (measured in mm^2) [173].

In this chapter, we use feature attributes to add cost and value information

per feature. This information, together with value and cost *functions*, are used to automatically prune and prioritize the products to be tested as described in Sections §6.3.3 and §6.3.4.

Adding cost data → In order to optimize the number of tests executed, we use feature attributes to store the cost of the resources needed to run each test. Estimating testing costs requires both understanding the resource needs of the tests (e.g. in order to rent virtual machine instances in the cloud), as well as understanding the domain-specific properties (e.g. pricing model of the cloud provider for using those resources and time required by each test).

Depending upon the product under test and tests being run, the test can consume more or fewer resources, such as the number of required virtual machine instances or the number of cores per instance. These resources are directly linked to the financial cost to run the tests on the target platform configurations. Each product is unique, and each feature may have complex non-linear interactions with other features that affect performance and, consequently, the resources needed to test it. Therefore, extrapolating the resources usage of a subset of configurations, such as all configurations that use Android 1.6, by manually calculating the value of a few fully specific configurations, such as {Android 1.6, screen size 240x320, 16MB RAM, has 'volume' hardware buttons, does not have 'Home' hardware button, etc} is difficult.

As stated in the previous sections, its expensive to execute every test on every mobile platform configuration. Therefore, it is necessary to select the mobile platform configurations that are worth testing. Knowing the cost of executing tests enables developers to evaluate the worthiness of executing those tests. Although appealing, the idea of adding value information may be considered infeasible in practice. However, we present a number of examples where test value can be practically determined (see Sections §6.2 and §13.3). For instance, testing an Android application that is running on an Android emulator will consume varying hardware resources, such as CPU, depending upon the emulator configuration.

When testing Android applications, users may want to test on as many valid hardware configurations (a.k.a emulator options) as possible. However, the number of potential emulator and device variations makes exhaustive testing infeasible. Further, because many organizations have limited development budgets, is important to provide mechanisms to maximize hardware coverage, in terms of feature coverage, given bounded testing resources.

For example, if we execute the Facebook Android application in front of five thousands different machines, and lately we need to execute as much test we can but with the half time, probably we will remove of the execution set

some of the configurations. Those metrics will allow us to select which tests to discard. When using a cloud based testbed this problem manifests as we do not know how the group of tests that keep bounded users costs. The fact is that the amount of available options to select or deselect when testing android applications makes hard to find the combination of those that minimizes the emissions or the costs. Using the ATAACK cloud enables users to test their applications in front of a large number of different hardware configurations. However, test an application in 32^7 options is expensive. To solve that an automated selection of the configurations that maximizes the hardware coverage is required.

Adding value data $\rightarrow$ Estimating the return on investment by executing a test is a difficult task. However, there are a number of scenarios where the benefits of individual tests can be derived. For example, when testing on mobile devices, the device market-share covered by the specified emulator configuration of the test can be used to estimate value. The greater the market-share associated with the platform configuration being tested, the more important the test is to execute.

An important attribute of the market-share driven approach to selecting mobile platform configurations to test on top of is that this approach can optimize the selection of platform configurations *with respect to the market-share coverage of the features that the product contains*. For example, if the five best selling Android devices are chosen for testing, these devices may be newer and have similar features to one another, such as large screens, that are not characteristic of the large installed base of previous generation phones. If screen size is a feature that has a direct impact on the product being tested, limiting the diversity of the screen sizes tested is not beneficial. Moreover, the product being tested will have a high likelihood of being installed on an older model device with a smaller screen size that it has not been tested on.

Value data may vary depending on external factors, such as new customers, new hardware requirements, or changes in the application design. TESALIA uses attributes to specify the value of testing each individual feature because they allow developers to update the value of each feature easily.

Defining the cost and value function $\rightarrow$ Referring to the motivating scenario presented in Section §6.2, an Android emulator with an extra high pixel density (high dots per inch [dpi]) screen uses substantially more processing power on the host computer than an emulator with a low pixel density (low dpi) screen. However, configuration options cannot be considered independently. For example, two hdpi (high number of dots per inch) screen emulators with different Android versions (e.g. 1.6 and 4.0) will have vastly different CPU utilization due to improvements in the operating system, and it is pos-

sible for an Android 4.0 hdpi emulator to use less CPU on the host computer than an Android 1.6 ldpi emulator. Without running all possible emulator configurations, it is hard to determine how much hardware, such as CPU, each platform configuration will require on a specific Android emulator configuration.

Defining the value of each test is also a key step when trying to optimize the set of tests to run for a software product-line. Again, TESALIA allows users to define the function to obtain the value of testing each product. In the example presented in Figure §6.4, the function $\sum$ *Audio.value, Wifi.value, Bluetooth.value*, is used as the value of each product (for the sake of simplicity).

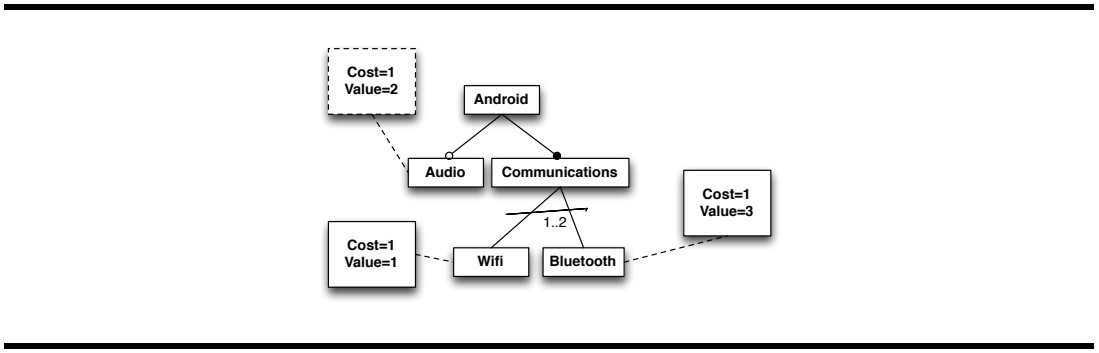


Figure 6.4: *Example based on the smartphone's motivating scenario.*

Figure §6.4 shows the example feature model with cost and value data on its attributes.

6.3.3 Pruning the testing scope

Testing on a large set of different product configurations can be very expensive, as shown in Section §6.2. When trying to reduce the number of tests, some products cannot be tested. To discard products when testing a product-line, the majority of approaches described in prior literature apply t-wise coverage to each feature sub-set [79, 80, 87, 98, 125, 130, 131]. T-wise testing consists of testing all products that contain the unique possible combinations of the selected *t* features. For example, if we plan to apply 2-wise (t=2) testing on the feature model presented in Figure §6.4, every combination of tests involving unique combinations two features in the product-line must be tested. On the other hand, if we plan to test by using 1-wise testing, every feature should be in at least one tested product.

When using TESALIA, any function can be used to define product testing value, since TESALIA is based on an underlying CSP solver. TESALIA can use the value of the cost attribute defined in each feature to obtain the cost of the product or can use the cost value to simply detect if a feature is present and then apply any t-wise based function to obtain the subset of products to be tested.

In order to automatically prune products for testing, TESALIA applies prior work on automated CSP-based derivation of feature model configurations to derive configurations sets (*e.g.*, mobile platform configurations to test on). Moreover, the configuration sets optimize a function over the features selected for testing without exceeding a testing cost budget(*e.g.*, cloud computing related costs). Once these products have been derived from the feature model, standard testing tools, such as JUnit, can be used to deploy and test a product in the cloud on the derived platform configurations.

The first step in deriving a set of products is to be able to derive all products from the feature model that do not exceed the testing budget. TESALIA builds on prior work for automated derivation of feature model configurations using CSPs to derive each individual platform configuration [19]. Note that we present the pruning of feature models as a new automated analysis operation extending the 30 existing approaches in the software product line community [19]. Given a feature model, a set of feature costs, a function to obtain the product cost and a maximum cost, the pruning operation returns a set of products that does not exceed the maximum cost provided. The TESALIA CSP model for deriving a single platform configuration from the feature model to test on top of is constructed as follows:

CSP model used by TESALIA to prune the products that exceed the maximum budget:

$$TESALIA =< F, FC, A, AC, f(AC), CC >$$

where:

- F is a set of variables, $f_i \in F$, representing the selection state of each feature in the product-line feature model. If the i_{th} feature is selected for a product test, then $f_i = 1$ and, otherwise, $f_i = 0$.
- FC is the set of constraints that define the different relationships between different features (e.g. if the i_{th} feature is a mandatory child feature of the j_{th} feature, then $f_i <=> f_j$) according to the mapping presented by [19].
- A is the set of attributes describing the estimated costs required to include each feature as part of a test product, where $a_i \in A$ is the cost to

include the i_{th} feature in a product to be tested. Note that depending on the testing scope the cost can be defined in different ways (*e.g.*, cost to execute the test in a machine, cloud related cost for testing, ...).

- $ac_i \in AC$ is a variable describing the current cost contributed by the i_{th} feature to the currently selected product. The cost of a feature is only incurred if it is selected, thus: $ac_i = f_i * a_i$.

- $f(AC)$ is a function that calculates the total estimated cost of testing the currently selected product. Several different functions can be used depending on the objective to maximize. For example, if we use $\sum_{i=0}^{n} ac_i$ to calculate the sum of each feature cost in a product we can specify not to exceed a concrete budget.

- CC is the maximum allowed cost for testing a product, thus: $f(AC) \leq CC$.

In order to ensure that only valid products are derived, a few additional constraints must be added. The set of variables, F, is used to describe the selection state of each feature. Initially, the root feature, f_0 of the feature model is labeled with value 1 (*e.g.*, indicating that the root feature is selected). Because the product being tested will typically require a subset of the features, such as minimum platform version or a rear-facing camera, these additional constraints are encoded into the CSP by labeling the corresponding variable f_i of a required feature with value 1 and the corresponding variable f_k of a deselected feature with value 0. Pre-labeling these feature variables removes them from consideration by the constraint solver and reduces the configuration space size. In other words, is possible to start with a partial configuration that contains the basics for making our product meet a minimum requirements, therefore reducing the computing power required by the CSP solver.

Once the initial labeling of the CSP has been performed, a constraint solver is used to derive values for the remaining variables in F, such that none of the feature model constraints encoded in the CSP are violated (an extensive discussion of encoding feature model constraints in CSPs is available in [19]). Moreover, the CSP includes the additional constraint that the total cost of testing on top of the selected platform configuration, described by the labeled variables in F, does not exceed the testing budget CC. The result of this automated labeling process is that the constraint solver produces a set of values for the variables in F, where the features that have been selected for the platform configuration are labeled with value 1 (*e.g.* if the feature f_i is selected then $f_i = 1$). The variables in F that are labeled with value 1 represent the configuration that has been derived for testing.

Applying this pruning method to our model presented in Figure §6.2 requires transforming the feature model into a CSP by using the following assignations of variables.

- F is the set of variables composed by Android, Communication, Audio, Wifi, Bluetooth, and Product.
- FC is the set of constraints representing the following relationships, Android **mandatory** Communication, Android **optional** Audio, Communication **set with cardinality 1..2** Wifi and Bluetooth.
- A is the set of attributes describing the estimated costs, in this example the features Audio, Wifi and Bluetooth have a cost of one $ per each one. For the sake of simplicity we are using dollars as a cost unit, however, other metrics such as CO2 emissions can be used.
- $f(AC)$ for this example we are using the sum of cost as a function to obtain the total cost of a product.
- CC is the maximum allowed cost, for the illustration purposed we fixed it to 3.

  This CSP returns all products having a $ cost less than 3, thus, filtering the products to test. In Table §6.2 all products with their cost are presented. Note that the gray line are products not returned by the TESALIA solution.

Name	Android	Communication	Audio	Wifi	Bluetooth	Product Cost
P1	✓	✓	✓	✓		2
P2	✓	✓	✓		✓	2
~~P3~~	~~✓~~	~~✓~~	~~✓~~	~~✓~~	~~✓~~	~~3~~
P4	✓	✓		✓		1
P5	✓	✓			✓	1
P6	✓	✓		✓	✓	2

Table 6.2: *Sub-set of products applying the function from Section §6.3.3.*

6.3.4 Prioritizing the tests list

There are circumstances where knowing which tests fit a bounded cost is not enough and it is also important to prioritize the set of tests to execute. Having tests prioritized, allows testers to begin with the tests that provide more

value, and thus, obtain more testing benefit in less time. TESALIA uses the value-attribute of each feature and a function to determine how much benefit is obtained by testing a product.

TESALIA requires developers to define a function to calculate the value of each product. This function should be adapted to each testing context. For example, if we want to value tests based on the number of features involved in a product, the value function would be $\sum f_i$. If we want to increase the market-share coverage of our tests we should attribute the feature model with market-share information and, thus, specify our function as $\sum value_i$ where $value_i$ is the value attribute of a feature present in the product.

CSP model used by TESALIA to prioritize the set of products to test:

$$TESALIAPrioritization =< V, VA, VAC, f(VAC) >$$

where:

- V is a set of variables and constraints presented in Section §6.3.3.
- VA is the set of attributes showing the estimated value obtained by including or not, each feature as part of a product, where $va_i \in VA$ is the value obtained by adding the feature i in a product to be tested. Different value functions may have more or less sense depending on the testing scenario we were working on.
- $vavar_i \in VAC$ is a variable describing the current value contributed by the i_{th} feature to the currently selected product. The benefits of having a feature on a product only appears if it is selected, thus: $vavar_i = f_i * va_i$.
- $f(VAC)$ is a function that calculates the total estimated benefits obtained by testing the currently selected product. Again, different functions can be used to prioritize the test list. For example, if we decided to value our test by a market-share based function, the value of each product can be represented by the product of the relative market-share of each feature.

Going back to the small example presented in Figure §6.2, and applying the CSP presented in this section we obtain a pruned list of products with a total value. Table §6.3 shows the list of products with its value attached. For doing so, we need to use the following values for each problem component.

- V is a the set of variables used to prune the solution space that ended by returning the set of products presented in Table §6.2.
- VA are the value attributes for each feature as shown in Figure §2.2.

- $vavar_i \in VAC$ is a variable that contains the relative benefit that is provided by testing the i_{th} feature. The benefits of having a feature on a product only appears if it is selected, thus: $vavar_i = f_i * va_i$. In this example we are using market-share coefficients as values for the tests.
- $f(VAC)$ is a function that calculates the total estimated value obtained by testing the currently selected product. For the sake of simplicity, we are using integers in the small example presented in Figure §6.2, therefore we can simply use the $\sum$ function.

Name	Android	Communication	Audio	Wifi	Bluetooth	Product Cost	Product Value
P1	✓	✓	✓	✓		2	3
P2	✓	✓	✓		✓	2	5
P4	✓	✓		✓		1	1
P5	✓	✓			✓	1	3
P6	✓	✓		✓	✓	2	4

Table 6.3: *Sub-set of products with testing value.*

For example, the following prioritization maximizes the value of the tested configurations: P2, P6, [P1,P5], P4. Note that the products P1 and P5 can both be executed in any order because they have the same assigned priority. By executing the products in this order, the tester knows that if testing is interrupted for some reason, the tests that will have been run are the highest value tests.

6.3.5 Packaging the most profitable set of products

After pruning and prioritizing the sets of product that meets budgetary and product-line constraints, being able to automatically derive the group of tests that maximize the value while keeping the costs bounded to a maximum cost is appealing. Note that the implementation of this knapsack is not multi-objective as the maximum cost is fixed. Thus, this solution will return the best value possible for a concrete maximum cost. Also, this operation requires both value and cost functions to calculate the weight and value of each element (product) in the knapsack.

For example, suppose that TESALIA has pruned the sets of possible configurations that do not exceed a 20$ cost and TESALIA also provided the value

of testing each product. In such as situation, how can we maximize the testing value of a 30$ cloud testing budget? To derive multiple products, TESALIA starts by discarding those products that exceed the total budget by themselves. This reduces the computational effort to build the product sets. Later, each derived product is represented as a new unique labeling of the variables in F. Once this set is derived, TESALIA constructs a knapsack problem to select the optimal subset of these mobile platform configurations to test on top of to maximize market-share

The knapsack problem is a classic problem in computer science. Several studies have proved that it is an NP-complete problem [40, 110]. In TESALIA we are using the 0,1 approach based on dynamic programming to solve it [110]. However, heuristics methods have been proposed to reduce the computational complexity of solving knapsack problems, such as the modified heuristic (M-HEU) algorithm [6]. The formal definition of the problem in TESALIA is defined as follows:

$$TESALIAmulti =< Sp, Cp, CMax, f(VAC) >$$

where:

- Sp is a set of variables describing the selection state of each possible configuration that could be tested. If the i_{th} possible product in P is selected for testing, then $Sp_i = 1$, if it is deselected then $Sp_i = 0$, where $Sp_i \in SP$. Sp define the set of elements to introduce in the knapsack. This is used to discard those products that exceed the maximum allowed budget by themselves.
- Cp is the cost of executing a test on top of the i_{th} platform configuration in P (*e.g.*, the derived result of $f(AC)$ from the previously described pruning step). This cost $f(AC)$ represents the volume in the knapsack problem.
- $CMax$ is the maximum allowed cost of executing the set of tests on the selected platform configurations. $CMax$ represent the maximum volume in the knapsack.
- $f(VAC)$ is the set of values per each product that represents the value of testing our product. As described in previous sections, one concrete metric for determining the value of a product is by using the market share it represents.

The goal of using this knapsack problem formulation is to derive the set of platform configurations that maximize the value of the test-suite being tested.

This objective function is encoded as:

$$Maximize(\sum_{i=0}^{n} Sp_i * f(VAC)_i)$$

Nevertheless, because developers have a limited testing budget, an additional constraint to bound testing cost should be considered:

$$CMax \geq \sum_{i=0}^{n} Sp_i * f(VAC)_i$$

Referring back to our simplified example presented in Figure §6.4, the results of deriving a set of products to test that fit within a budget of 6 cost units are shown in Table §6.4. The best combination of products to test for the given budget is captured in *G3*.

Group Name	Products set	Set Cost	Set Value
~~G1~~	~~{P1,P2,P4,P5}~~	~~6~~	~~12~~
~~G2~~	~~{P1,P2,P6}~~	~~6~~	~~12~~
G3	{P2,P4,P5,P6}	6	13

Table 6.4: *Optimized test products for a budget of 6 cost units.*

6.4 Summary

In this chapter we present the lessons we learned while developing the TESALIA solution. A key challenge when testing a software product line is the large number of different products that can be encoded into it. In the mobile development industry, a similar problem exists, called platform fragmentation, where developers must deal with different features such as major versions of the mobile OS or varying screen sizes. In most cases, testing all possible platform configurations is not feasible, and developers therefore select a set of popular phones to test on. However, no studies have been performed to determine the effectiveness of this approach over alternative approaches.

This chapter presents TESALIA, a framework for selecting SPL products (e.g. complete device descriptions) to test applications on. TESALIA extends

prior work on automated analysis of feature models by deriving which products, or groups of products, an application should be tested on to provide the most value. Value is defined by a series of cost functions provided by the developers. We provide an example cost function that balances the desire to test software on platform feature configurations that cover the largest percentage of the mobile market (e.g. the current market) and the desire to keep testing costs low.

Chapter 7

Testing: pair-wise pruning optimization operations

A grands maux, grands remèdes.

Dicton populaire, Frenchmen

Testing variability intensive systems introduce new challenges when using feature models such as the needing of coping with continuous variables using floats or integers. In this chapter, we present VANE, a variability-based testing approach to derive pair wise covering sets that optimize certain values. VANE computes T-wise covering sets while optimizing a function over attributes. Moreover, VANE is able to find pareto optimal solutions by using an "a proiri" solution. [†1].

[†1]This chapter is based in [69] and part of this material has been published in the ISSTA conference

7.1 Introduction

Variability intensive systems are becoming more and more common in our society. However, the complex scenarios described in variability intensive systems variability models, usually introduces new constructs that require new automated analysis mechanisms. For example, when describing the variability of video analysis systems [126, 138], part of the variability comes encoded in float and reals which represent realistic information. Moreover, decisions over variability intensive systems are not usually taken by only one practitioner but more than one.

In the previous chapter, we propose the use of automated analysis for pruning, prioritizing and packaging feature model products. However, the new constructs introduced by variability intensive systems make interesting to apply pruning and prioritization at the same time. For example, to obtain a set of products that while meeting a certain covering criteria optimize a certain function over quality attributes.

In this chapter, we present VANE, a variability-based testing approach for deriving optimal –based on quality attributes– t-wise covering sets. These configurations can be exploited afterwards to test variability intensive systems. In this research we rely in feature models [19, 22, 136] which are the most popular notation for modeling and reasoning about variability. We use advanced constructs such as attributes for handling numerical parameters and preferences. We apply combinatorial testing [86, 87, 98, 125, 130, 131] over feature models with attributes to reduce the number of configurations (combinations of features and attributes).

VANE is a hybrid approach mixing constraint satisfaction problem (CSP) solving techniques and evolutionary algorithms. The CSP is used to obtain T-wise covering sets while the genetic algorithm is used to tackle the multi-objective nature of the problem. A unique property of VANE is that it can obtain the minimal T-wise coverage while optimizing a function over attributes, for example, to minimize a custom attribute such as the video luminance.

Previous research proposed to use different metrics to optimize test-suites for concrete users needs in variability-intensive systems [81, 86, 130, 131]. These approaches allowed assigning more importance to some inputs than others when testing. However, they only focused on functional testing of the main system features without considering different testing objectives including quality *attributes*. Other evolutionary-based approaches do not consider testing aspects [151].

We have created a solution to address the problem of optimizing a T-wise covering set while taking into account quality attributes and functional information. VANE relies in two artificial intelligence paradigms and takes as input a feature model describing an variability intensive system. First, VANE builds a constraint satisfaction problem (CSP) to obtain t-wise covering sets that encode all t-wise covering sets for a concrete feature model. This CSP also enables users to optimize custom objective functions such as minimizing the cost of a test-suite. Later, VANE uses an evolutionary algorithm to optimize more than one function at the same time. For example, to obtain the minimal t-wise coverage (cover all pairs using a minimal amount of input configurations) and maximize a custom quality attribute such as the cost of a test.

In this chapter we provide the following contributions:

- An original motivating scenario for variability and testing techniques to the domain of video sequence analysis, in the context of an industrial project.
- CSP encoding and automated techniques to grant T-wise coverage for feature models with attributes. We also develop multi-objective solving techniques to obtain T-wise configurations sets that satisfy multiple testing criteria at the same time.

7.2 Motivating scenario

Video analysis systems are ubiquitous and crucial in modern society [126, 138]. Their applications range from video protection and crisis monitoring to crowds analysis. *Video sequences* are acquired, processed and analyzed to produce numerical or symbolic information. The corresponding information typically raises alerts to human observers in case of interesting situations or events. For instance, a classical scenario in natural disasters is to recognize survivors by using airborne cameras with the intention of rapidly acting based on information gleaned to achieve strategic or tactical rescue.

Depending on the goal of the video sequence recognition, signal processing algorithms are assembled in different ways. Also, each algorithm is a complex piece of software, specialized in a specific task (e.g., segmentation and object recognition). Even for a specific job, it is difficult to find a one-size-fits-all algorithm capable of being efficient and accurate in all settings. The engineering of video sequence analysis systems, thus, requires choosing and configuring the right combination of algorithms [138].

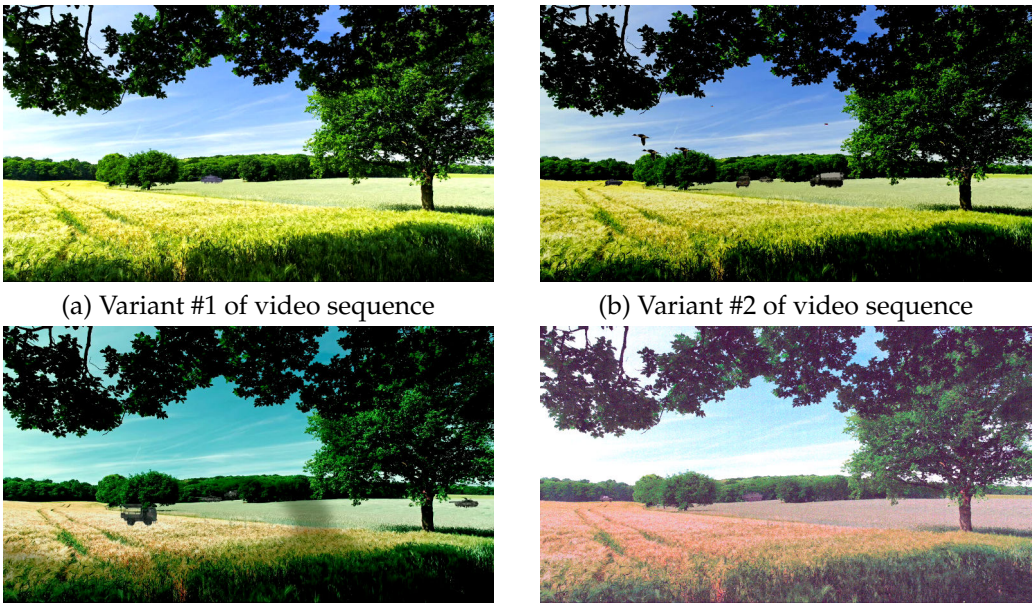
In practice, engineering such systems is an iterative process in which algorithms are combined and tested on diverse inputs (video sequences). Practitioners can eventually determine what algorithms are likely to fail or excel in certain conditions before the actual deployment in realistic settings. Admittedly, practitioners rely on empirical and statistical methods, based on numerous metrics. However, the major barrier to improve analysis algorithms is to find a *suitable and comprehensive input set of video sequences for testing analysis algorithms*. The current testing practice is rather manual, very costly in time and resources needed, without any qualitative assurance (e.g., test coverage) of the inputs [122, 188].

In a project involving three industrial partners, we observed that collecting videos for testing such algorithms is difficult. The targeted scenarios should challenge algorithms to process video sequences by introducing high *variability*, for example, different kinds of luminosity, altitudes, instability, meteorological conditions, etc. By combining the different variation points of the video sequences, we identified 153000 possible video sequences (three minutes each), corresponding to 320 days of videos to process and sixty-four years of filming outdoors (two hours per day). The numbers were calculated at the beginning of the project and the situation is now worth with billions of possible video sequences. These values show that the current practice, based on a manual elaboration of video sequences, is too expensive in terms of time and resources needed. Moreover, a related problem is that practitioners ignore what kinds of situations are covered or not by the set of video sequences.

7.2.1 Variability-based testing approach

To overcome the previous limitations, we introduce a generative approach, based on variability modeling and testing principles. The goal of the approach is to automatically synthesize a variant of a video sequence given a configuration (i.e., a selection of desired features). Compared to the current practice, the approach aims to provide more automation, more diversification and more control when collecting input video sequences.

For example, we synthesized four different variants of video sequences (see Figure §7.1). The first variant is a video sequence with a very intense luminosity and a tank moving on. The second variant differs from the first variant: some birds and other kinds of vehicles are included while the contrast is more intense. Variant #3 introduces shadows to mimic passing clouds. Also, Variant #4 is over expose, thus, some colors are hardly distinguishable. We only describe static parts of the video sequence variants but the dynamic parts is impacted as well (e.g., motion of vegetation due to the wind, appear-



(a) Variant #1 of video sequence

(b) Variant #2 of video sequence

(c) Variant #3 of video sequence

(d) Variant #4 of video sequence

Figure 7.1: *Four variants of video sequences.*

ance of occultants, vibrations of camera, or shadows). Eventually, much more variants than the four depicted in Figure §7.1 can be synthesized to test computer vision algorithms (e.g., an algorithm in charge of tracking vehicles) in diverse and challenging settings.

As part of the approach, *variability modeling* is used to formally characterize what can vary in a video sequence and delimit the relevant testable configurations. Because of the huge number of testable configurations, *combinatorial testing* techniques are applied to obtain the minimal T-wise coverage while optimizing attributes. An overview of the approach (in the context of the MOTIV project) is given in Figure §7.2. At the starting point (see the top of the figure), a variability model is elaborated and characterizes a set of configurations (see next section for more details about the so-called feature model with attributes). Testing techniques (see Figure §7.1.a) operate over the model and are in charge of producing a relevant subset of video sequences. A transformation (Figure §7.1.b) has been developed to obtain configuration files that consists on variables and values. Lua code developed by one of the MOTIV partner, processes the configuration files and executes some algorithms to alter, add, remove or substitute elements in a base video sequence. We obtain at the end variants of video sequences (Figure §7.1.c). Note that, Lua is a widely

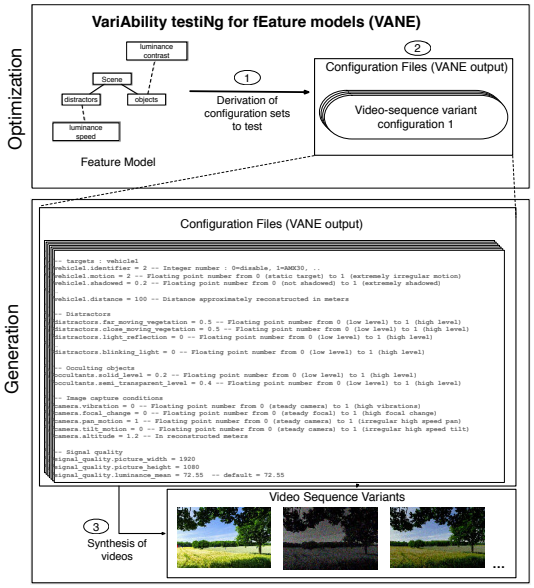


Figure 7.2: *Testing in the MOTIV project.*

used programming language (http://www.lua.org/). Details about the vision algorithms in charge of synthesizing video sequences' variants are out of the scope of the thesis.

7.3 VANE solution

Despite the constraints over attributes and features, the number of possible configurations is enormous. Exhaustive testing in such a large space of configurations is clearly unfeasible. Literature in the past proved that most errors can be detected when using pair-wise combinations of inputs [172]. Moreover, Cohen et. al. [41] proved that those results apply to feature models. Our approach is to test configurations of video sequences that cover all possible T feature interactions (*T-wise*). In theory, T-wise dramatically reduces the number of testable video sequences while ensuring reasonable coverage.

VariAbility testiNg for fEature models (VANE) is a solution to obtain T-wise covering sets for feature models with attributes. VANE follows a set of steps to obtain T-wise covering sets of configurations while meeting different user criteria (e.g. minimize the cost of a set of configurations). Figure §7.3 shows the VANE process. First, developers encode the variability intensive

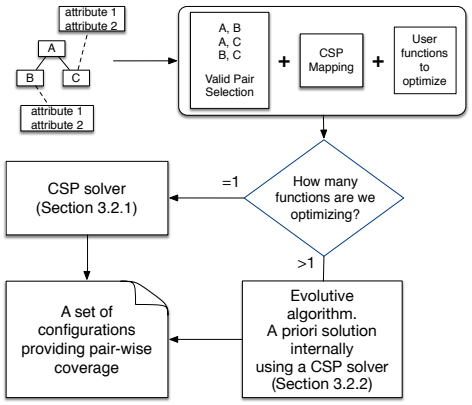


Figure 7.3: *VANE process to obtain optimal T-wise covering sets.*

system's variability using an attributed feature model. Second, VANE obtains the valid permutations of features to be covered. Third, VANE encodes the input model as a CSP. Later, VANE adds different constraints to the CSP depending on user requirements.

In the case that the user wants to obtain a multi-objective solution, VANE implements an "a priori" solution by using a genetic algorithm. This solution uses the previously generated CSP to find the weights that return Pareto optimal solutions.

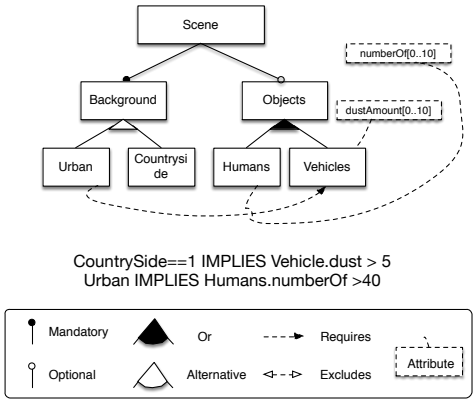


Figure 7.4: *An exemplified feature model with attributes.*

7.3.1 T-wise CSP for attributed feature models

This section describes how VANE uses CSP to derive solutions for T-wise covering arrays. Prior work in the field of automated analysis of feature models achieved to extract information from feature models by using computed-aided mechanisms. Those works yielded a set of different operations and translations into CSP problems [19].

In this chapter, we consider the derivation of T-wise covering sets as an automated analysis operation that takes as input attributed feature models and user preferences. After a CSP formulation is defined for obtaining T-wise configuration sets, VANE can derive all the different valid combinations of configurations that fully covers a set of feature pairs.

7.3.2 T-wise covering sets for attributed feature models

VANE can reason over all possible configuration sets that cover a concrete set of T-wise combinations. For example, in the case of the model presented in Figure §7.4, VANE retrieves the configuration set covering all feature combinations such as Urban and Humans. To cover a set of feature combinations, VANE uses a custom mapping between feature models and CSP. The mapping used is defined by the tuple:

$$< P, F, FC, A, AC, PC >$$

where:

P is the set of feature combinations to be covered. P_{ij} represents the feature j of the feature combination i needed to be covered by a configuration in the test-suite.

F is a set of variables representing the features in the feature model. If the variable f_i is equal to 1, then the feature F_i is going to be present in the configuration. (e.g if the feature i is a mandatory child of the feature j, the constraint $f_j \Leftrightarrow f_i$ is in this set)

FC is the set of constraints representing the different relationships between the model elements. This is between different features and between features and attributes.

A is the set of variables representing the different attributes existing in the model.

AC is the set of constraints between different attributes. For example, is the cost should be greater than 40 a constraint representing that will be added.

PC is the set of constraints representing the constraints granting the coverage of each pair. This is, for each pair P_{ij}, the constraint $F_j = 1 \wedge F_i = 1$ is introduced in the CSP.

This mapping differs from previous approaches because it is intended to derive combinations of valid configurations (covering sets) instead of single configurations. Table §7.1 shows the main differences between the previous mapping for single configuration derivation proposed by Benavides et al. [19] and the one used by VANE. Note that in the table, F^P represents the parent feature of the relation. F^{C1} to F^{CX} represent the children features of a relation where X is the *nth* child of the relation.

7.3.3 T-wise covering sets optimization

There is more than one solution for the problem of finding T-wise covering sets. Moreover, some covering sets may fit better concrete user preferences. For example, some users may be interested in reducing the number of configurations to run while others prefer to reduce the time to test or other quality attributes of the test-suite. VANE enables users to maximize a concrete function over features and attributes by adding constraints into the CSP.

Optimizing a function over features and attributes. VANE enables T-wise practitioners to decide which function between attributes and features fits better the user desires.

More complex constraints are also allowed when using VANE. For example, if the user wants to minimize the number of different features when generating the test suite, we need to fix the value of the function to a variable:

$$\mathtt{DifferentNumberOfFeatures} \sum_{i=1,\, j=1}^{n\ m} f_{ij}$$

where *n* is the number of pairs to cover and *m* the number of features in the feature model.

Minimizing the number of different configurations to use in a T-wise covering set. It is interesting to use as few configurations as possible when performing T-wise coverage. This optimization is known as the minimal or optimal T-wise [101] and different constraints should be introduced in the CSP to obtain it.

Relation	Traditional mapping	VANE mapping
Mandatory	$F^P = F^{C1}$	$\forall P_{ij}$ in P , $F^P_{ij} = F^{C1}_{ij}$
Optional	$if(F^P = 0)$ then $(F^C = 0)$	$\forall P_{ij}$ in P, $if(F^P_{ij} = 0)$ then $(F^C_{ij} = 0)$
Or	$if(F^P = 1)$ then $\sum(F^{C1}, F^{C2}, ..., F^{CX})$ in 1..X else $\sum(F^{C1}, F^{C2}, ..., F^{CX})$	$\forall P_{ij}$ in P, $if(F^P = 1)$ then $\sum(F^{C1}_{ij}, F^{C2}_{ij}, ..., F^{CX}_{ij})$ in 1...X else $\sum(F^{C1}_{ij}, F^{C2}_{ij}, ..., F^{CX}_{ij}) = 0$
Alternative	$if(F^P = 1)$ then $\sum(F^{C1}, F^{C2}, ..., F^{CX}) = 1$ else $\sum(F^{C1}, F^{C2}, ..., F^{CX}) = 0$	$\forall P_{ij}$ in P, $if(F^P_{ij} = 1)$ then $\sum(F^{C1}_{ij}, F^{C2}_{ij}, ..., F^{CX}_{ij}) = 1$ else $\sum(F^{C1}_{ij}, F^{C2}_{ij}, ..., F^{CX}_{ij}) = 0$
Excludes	$if(F^P > 0)$ then $(F^{C1} = 0)$	$\forall P_{ij}$ in P, $if\ (F^P_{ij} > 0)$ then $(F^{C1}_{ij} = 0)$
Requires	$if(F^P > 0)$ then $(F^{C1} = 1)$	$\forall P_{ij}$ in P, $if\ (F^P_{ij} > 0)$ then $(F^{C1}_{ij} = 1)$
Link between features and attributes	Not required	$\forall P_{ij}$ in P, $if\ (F_{ij} > 0)$ then $Attr_{ij} = value$
Complex Constraints	Not required	These constraints will be mapped depending on the constrain itself
Pair-wise constraints	Not required	$\forall P_{ij}, F_j = 1 \wedge F_i = 1$

Table 7.1: *Single derivation CSP vs T-wise covering set derivation CSP.*

First, we need to introduce a set of reified variables that represents if the configuration covering the pair *i* is different from the configuration covering the pair *j*.

$$if\ (P_i \neq P_j)\ then\ reified_i = 1$$

Later, we minimize the sum of reified variables, thus, minimizing the number of different configurations used.

$$\min \sum_{i=1}^{n} reified_i$$

For example, the pair-wise combinations existing in the model presented in Figure §7.4 can be covered by the configurations i) mobile platform fea-

ture"Scene, Background, Objects, Humans, Vehicles, Urban" and; ii) mobile platform feature"Scene, Background, Objects, Humans, CountrySide". This is a minimal set of configurations as we cannot reduce the number of configurations while covering all pair-wise combinations.

7.3.4 Obtaining multi-objective test-suites

In our context, our goal is to generate T-wise covering sets while optimizing more than one objective function. For example, users might want to minimize the value of concrete attributes while still using as less as possible configurations. Note that optimizing different functions at the same time might not yield optimal values for them separately but a good trade-off between them. This problem is commonly known as multi-objective optimization problem [51].

Our initial experiments showed that exact solutions hardly scale when having complex attributes (see section §10.2). Therefore, we use an *"a priori"* solution based on genetic algorithms [52]. In VANE approach, the "a priori" solution is based on a mix of CSPs and genetic algorithms for multi-optimization problems. The hybrid solution VANE internally uses the custom CSP-mapping presented in Section §7.3.1 to evaluate the fitness function.

When defining the multi-objective function for obtaining multi-objective T-wise covering sets, we define a new function to optimize:

$$F(x) = w_1 * F_1(x) + w_2 * F_2(x) + ... + w_k * F_k(x)$$

where k is the number of different functions to optimize and w represents the weights of each function to be determined by the genetic algorithm. Note that all functions should return values normalized between 0 and 1.

Later, we created a genetic algorithm that finds the values of the w_k values that correspond to each Pareto optimal value [51] of the function. The genetic algorithm is defined by:

- **Gen.** A gen in the algorithm is a float representing the weight of a concrete w_k.
- **Individual.** An individual is represented by a set of genes representing all w_k in the function to optimize. Therefore, an individual is a vector of floats representing weights for each function to optimize.
- **Crossover.** In this problem, the crossover operator is based in getting two random genes and switch their values.

- **Mutation.** The mutation operators increment or decrement a gen value in a random quantity. This quantity will affect the precision of the Pareto optimal found. But also, it will increase the search space.
- **Selection method.** There are several methods to select the best individuals from each generation. However, for the sake of simplicity, only ranked based selection methods are used in this problem.

Let us consider an example in Figure §7.4. If we want to optimize the attribute $dustAmount$ and minimize the number of different configurations for a 2-wise coverage, we should use the fitness function:

$$\max(w_1 * \sum_{i=1}^{n} dustAmount_i + w2 * \sum_{i=1}^{n} numberofEqualConstraints)$$

This will return the Pareto solution: mobile platform feature"Scene, Background, Objects, Humans, Vehicles, Urban".

7.4 Summary

We cannot test everything when coping with variability intensive systems. The domain of video analysis is not an exception to the rule: testing the computer vision algorithms under all combinations of inputs (video sequences) is not feasible. The manual elaboration of a test suite of variability intensive systems is a possible solution, but as reported in an industrial project, the process is very costly in time and resources. As a result, practitioners face severe difficulties to tests able of covering the diversity of targeted video analysis scenarios.

In this chapter, we have described an original approach combining variability and testing techniques. A formal variability model (i.e., feature model with attributes and multi-features) documents what can vary within a video sequence. Combinational and multi-objective optimization techniques have been presented to generate a certain number of configurations. The configurations are exploited afterwards to synthesize variants of video sequences. Specifically, our solution called VANE computes T-wise combinations of the parameters while maximizing a custom function over quality-attributes.

Chapter 8

Configuring while drifting feature models

I was like a summer baby.
But I did grow up in the South.
I grew up in serious, serious Appalachia, in a very small town.

William Gibson, Novelist

The increasing complexity and cost of software-intensive systems has led developers to seek ways of reusing software components across development projects. One approach to increasing software reusability is to develop a Software Product-line (SPL), which is a software architecture that can be reconfigured and reused across projects. Rather than developing software from scratch for a new project, a new configuration of the SPL is produced. It is hard, however, to find a configuration of an SPL that meets an arbitrary requirement set and does not violate any configuration constraints in the SPL. Existing research has focused on techniques that produce a configuration of an SPL in a single step. Budgetary constraints or other restrictions, however, may require multi-step configuration processes. For example, an aircraft manufacturer may want to produce a series of configurations of a plane over a span of years without exceeding a yearly budget to add features[†1].

[†1]Part of the material used in this chapter was accepted in the JSS journal[179]

8.1 Introduction

When software evolves, its evolution may need to be broken into multiple steps to satisfy evolution constraints [1]. In some cases, product features must be introduced gradually over a series of steps. For example, the Boeing 737 aircraft, introduced in 1966, has been continually upgraded and adapted over time and is still currently in service. Each successive configuration of the 737, which is called a *Variant* has been developed over multiple years and incorporated new features into the base aircraft configuration [159]. For example, development of the 737-300 configuration of the aircraft started in 1979 and first flew in 1984. The configuration added a variety of features, such as an Electronic Flight Instrumentation System. The 737 has had numerous successive configurations, such as the 737-400, 737-500, 737-600, 737-700, 737-800, and 737-900, all planned and developed over significant spans of time.

There are a number of scenarios where the evolution of a set of products may be performed over several predefined steps. For example, when a new Linux distribution such as an Ubuntu release is planned, developers have to decide the set of software artifacts that are going to be added and removed in the next release of the distribution (e.g. add and remove packages and change the dependencies between them). Moreover in other domains, such as aircraft construction, nuclear power plants, etc., configurations and upgrades to product configurations are planned years in advance (*e.g.* the configurations of the 737 have spanned 46 years) and must be analyzed years in advance of their actual production. Ideally, an aircraft manufacturer would like to derive a sequence of successive configurations that build upon one another, as the 737 variants do, so that more advanced features are included each year. A manufacturer, however, cannot arbitrarily choose features to add in a given year. Instead, each set of features for a year must constitute a complete and correct configuration of the SPL to avoid selling a defective and non-viable configuration.

Further complicating this scenario is that a manufacturer is constrained in its introduction of features. For example, a manufacturer must introduce features in an order that ensures no two successive configurations will not differ by more than the price that a customer is willing to pay from one year to the next (*e.g.*, airline development or acquisition budget). Therefore, not only should every configuration step should satisfy a set of constraints, but the delta between any two successive configurations must be also acceptable.

Finally, when the product life spans years, such as the case of the 46 year history of the 737, the availability and capabilities of the processors, software, sensors, and other constituent components of the product inevitably change.

Not only must manufacturers be able to plan and reason about configuration over multiple steps but have plans that account for the end-of-life of components and the significant increases in capabilities of newer components, which produce changes in the underlying feature model. For example, the processing power and availability of the processors used in the 737 have changed dramatically from 1966 to 2012. In some cases, the feature model may be specialized (*e.g.*, adapted so that its valid configurations at later steps are subsets of the starting set of valid configurations). In other cases, new features may be added to the feature model so that it is evolved to allow configurations that were not initially possible or valid. Thus, when the configuration problem should be reasoned in multiple steps, manufacturers must deal with two distinct forms of change: 1) changes to configuration and 2) changes to the underlying feature model that dictate what configurations are valid.

This process of producing a series of intermediate configurations between a starting configuration and a desired ending configuration—*i.e.*, a *configuration path*—is shown in Figure §8.1. In Figure §8.1, the selected features are colored in grey and the red features represent feature selections that violate a software evolution constraint. This sequence of activities is called

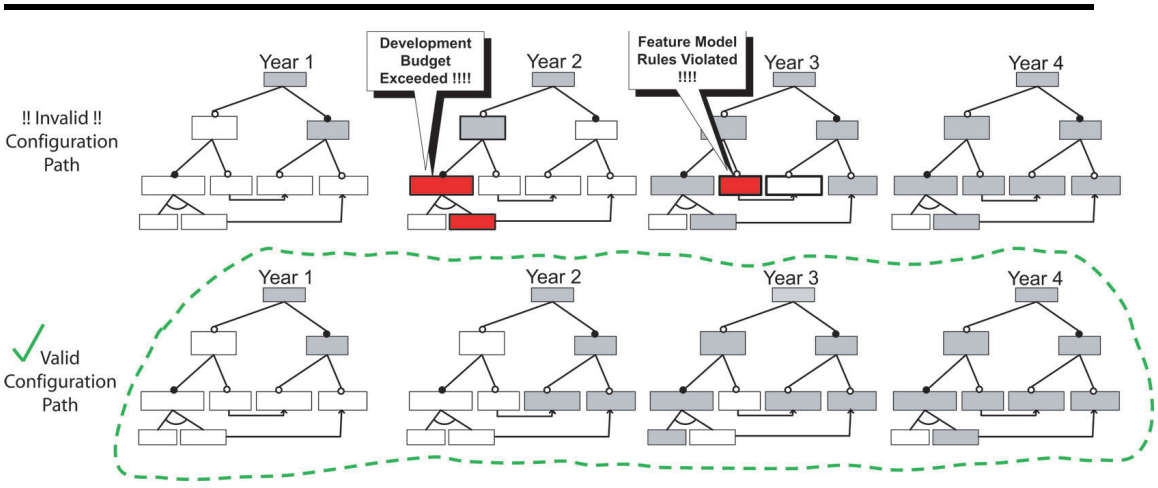


Figure 8.1: *Potential configuration paths.*

a *multi-step configuration problem*. Prior work on automated configuration [12, 25, 33, 108] focuses on selecting a single configuration in a single step and not determining a configuration path. As a result, developers must manually derive a configuration path through feature models with hundreds or thousands of features and complex constraints on how successive configurations can differ.

Manually deriving configuration paths for a product-line is hard because developers must analyze a myriad of tradeoffs related to the order that the features are selected. For example, developers may temporarily add a feature that is not in the desired ending configuration to yield a valid variant at a particular step. Moreover, the costs of introducing features may vary over different steps making it hard to identify exactly the right step to introduce a feature. For example, the cost of adding an entertainment system in an aircraft may vary from one year to the next one because of variations in display prices, fluctuations in currency value, or changes in tax rates.

8.2 Reasoning over multi-step configuration problems

We have developed an automated method for deriving a set of configurations that meet a series of requirements over a span of configuration steps. We call our technique the *MUlti-step Software Configuration probLEm Solver* (MUSCLES). MUSCLES transforms multi-step feature configuration problems into *Constraint Satisfaction Problems* (CSPs) [178]. Once a CSP has been produced for the problem, MUSCLES uses a constraint solver to generate a series of configurations that meet the multi-step constraints. MUSCLES can return either all valid paths or a single optimized path from the initial configuration to the final one and the SPL engineer can decide which evolution path best fits the project's goals.

This section presents a CSP model of the multi-step configuration approach used by MUSCLES to derive valid configuration paths of SPLs. This chapter also presents the techniques for modeling multi-step configuration problems as CSPs. These techniques give modeling tool developers the theoretical underpinnings to develop tools that can reason about configuration over multiple steps. We have developed domain-specific graphical modeling tools for our industry partners, using the Generic Eclipse Modeling System (http://eclipse.org/gmt/gems), for describing these problems and each of the various constraint types outlined in this chapter and automating the transformation to CSP. For example, in past work with Boeing, Siemens, and others, we have developed modeling tools on top of these types of algorithmic approaches for the aeronautics, automotive, and mobile computing domains [53, 59, 75, 107, 119, 181–183]. However, the process of building domain-specific languages and tooling on top of MUSCLE is beyond the scope of this chapter and focuses on the CSP modeling and solving process for these multi-step problems. SPL modeling experts can build modeling tools that use model

to model transformation to create MUSCLES CSPs, solve for optimized configuration paths, and present the results to the user, very similar to our work in [181].

This approach assumes that developers have advance knowledge of the feature model changes that will occur. In some cases, unforeseen changes may arise that impact the configuration paths that were previously derived with MUSCLES. Unforeseen changes are always a challenge in software development. A key attribute of MUSCLES is that it can help developers to quickly analyze a number of different configuration paths to understand the impact of an unforeseen change that has arisen. MUSCLES does not guarantee that the configuration paths may not change due to unforeseen circumstances, but it does help engineers to reason about how those changes may impact future configuration decisions and aid them in understanding corrective remedies.

In its most general form, multi-step configuration involves finding a sequence of at most K configurations that satisfy a series of point configuration constraints and edge constraints. This definition requires the start and end configurations meet a set of point constraints, but does not dictate that a *single* valid starting and ending configuration exist. All derived configurations at each step must be complete and valid feature model configurations – no partial configurations are allowed.

General formal model. We define a multi-step configuration problem using the 6-tuple $Msc =< E, PC, \Delta(F_T, F_U), K, F_{Start}, F_{end} >$, where:

- E is the set of edge constraints, such as the maximum development cost per year for features,
- PC is the set of point configuration constraints that must be met at each step, such as the feature model rules that developers may require to be adhered to across all steps (the point configuration constraints do not have to be identical across all steps. For example, if feature A is active in the step K is not required to explicitly set it to true in the step k+1),
- $\Delta(F_T, F_U)$ is a function that calculates the change cost or edge weight of moving from a configuration F_T at step T to a configuration F_U at step U,
- K is the maximum number of steps in the configuration problem,
- F_{Start} is a set of configuration constraints on the starting configuration. Those constraints can be a list of features that must initially be selected. For example, the basic security required when building an aircraft.
- F_{end} is a set of configuration constraints on the final configuration to be reached at the end of the configuration steps. Those constraints can be

composed by the features that must be selected or maximum cost of the final configuration. For example, the maximum cost for the entertainment system on an aircraft.

We define a configuration path from step T over K steps as a K-tuple

$$P =< F_T, F_{T+1}, \ldots F_{T+K-1} >$$

, where the configuration at step T is denoted by F_T. Each configuration, F_T, denotes the set of selected features at step T.

8.3 Modeling feature model drift

When configuration occurs over multiple steps, the configuration process may span a substantial period of time. For example, the aeronautics development example from Section §8.1, where features are being added to a plane, spans several years. In most multi-step configuration problems, such as the Boeing 787, developers may need to reason about configuration over a span of days, months, or years in order to decide the best path. For example the Boeing 787 exhibited a number of feature model drifts during its configuration. Figure §8.2 shows the feature model changes made to this aircraft since its first release [†2]. Figure §8.2 shows the changes that occurred over several years. In the period, from 2006 to 2014, the option of having 3 rows and 290 passengers was added. These improvements required the removal of the features "126.920L" and "3 rows/210 passengers" from the feature model, as well as the mandatory child relationship between "Seats Configuration" and "3 rows/210 passengers". Finally, two new features, "3 rows/290 passengers" and "138.700L," were added to the feature model. Further, the next planned release the plane in 2014 will remove the "3 rows/210 passengers" feature and the set relationship for seating and add the feature "3 rows/310 passengers" as the only seating option.

Configuration time frames that span months or years introduce the possibility for *feature model drift*. Feature model drift is the evolution of a feature model, through the addition or removal of features and model relationships, after the initial configuration step. The allowed changes depend on the expressiveness of the models being used. For example, if cardinality based feature models are used, the changes can also incorporate changes in the cardinalities of relationships. If attributed feature models are used, changes to the values of attributes and attribute relations can be captured [145]. For, example aircraft manufacturers may rely on suppliers that plan to introduce new features in

[†2] http://www.boeing.com/commercial/787family

a component at a specific time. Moreover, suppliers may plan to discontinue support for older features in the future.Note that, when a feature model drifts it is possible that errors may be introduced, such as contradictions that yield an unsatisfiable feature model. MUSCLES assumes that the feature models at each step are satisfiable and error-free. In this chapter we do not focus on this problem but it can be easily addressed by using the error checking techniques listed by Benavides et al.[19], and the parsers from FaMa[20], TVL[37] or SPLOT[112] for lexer errors.

When using MUSCLES to analyze feature model drift, SPL managers need to be able to predict the changes ahead of time in order to reason about them, which is not always possible. Sometimes the drift occurs due to security flaws detected in advanced stages of development or due to market requirements. MUSCLES is useful for reasoning about how these changes can or will impact planned configurations.

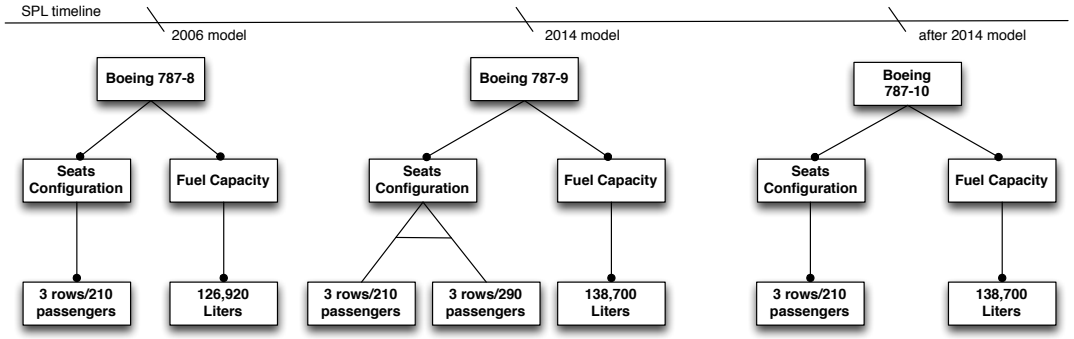


Figure 8.2: *Boeing 787 feature model drift.*

In many cases, developers do know ahead of time which features or relations will be introduced, discontinued, or replaced. Moreover, developers often have an estimate of when the availability of the feature (and its relationships with the rest of the model) will change based on information provided by a supplier or other mechanism. This data on feature and constraint modification times allows developers to incorporate this knowledge into the construction of a multi-step configuration problem. This section describes how feature model drift can be accounted for in a multi-step configuration CSP.

8.3.1 Modifying the CSP model of multiple steps

In the original formulation of the CSP, the set of features that are present does not change over time. To account for feature model drift, we show how we can relax our requirement from Section §8.2 that feature model constraints remain static. Once feature model constraint changes over multiple steps are modeled in the CSP, the solver can derive a configuration path that respects the feature model constraints as they drift. This eliminates the burden on developers to derive configuration paths that must meet complex drifting feature model requirements. An important point, however, is that this approach explicitly models the addition, removal of features and relationships in the future.

As we showed in Section §8.2, we constrain the feature selection variables F_T to respect the feature model constraints. Since each variable represents the selection state of a feature at a specific step, **we do not have to apply the same constraints to every step.** If the j_{th} feature is an optional child of the i_{th} feature (the software package) at step T and at step K, the j_{th} feature becomes mandatory, we can model this as:

$$(f_{jT} = 1) \quad \Rightarrow \quad (f_{iT} = 1)$$

At Step K, the j_{th} feature becomes mandatory, changing the constraints on selection of the feature:

$$(f_{iK} = 1) \quad \Rightarrow \quad (f_{jK} = 1)$$
$$(f_{jK} = 1) \quad \Rightarrow \quad (f_{iK} = 1)$$

That is, at step T, if f_i is selected ($f_{iT} = 1$) there is no constraint requiring f_j to be selected. At step K, however, there is the constrant that $(f_{iK} = 1) \Rightarrow (f_{jK} = 1)$, which makes f_j mandatory.

Examples of other feature model drifts as CSP constraints are shown in Figure §8.3.

The approach described above can handle arbitrary modifications to a feature model as long as the modifications yield a new feature model with at least one valid product. If a contradiction is introduced via feature model drift and no valid products are present, the solver will not be able to derive a configuration path. Another possible contradiction is if the edge or point configuration constraints contradict the changes introduced by feature model drift. For example, if a feature that is mandated by a point configuration constraint is removed by feature model drift, a contradiction occurs. The approach requires that neither type of contradiction be present.

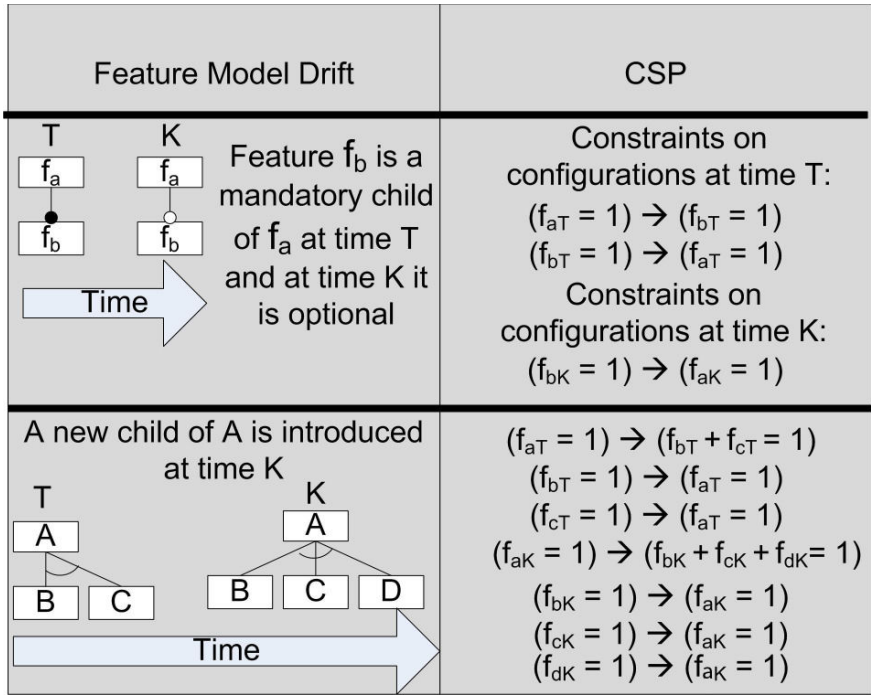


Figure 8.3: *A CSP model of feature model drift.*

8.3.2 Feature drift epochs

Because feature model drift may take place far in the future, it may not always be possible to precisely predict the time step at which a particular feature becomes available. For example, a supplier may indicate that in the next 3-5 years, they plan to phase out the usage of a particular component. In these scenarios, SPL engineers need a way to be able to reason about configuration and place bounds, rather than exact times, on feature model drift.

The formal model of feature model drift that we have presented can be extended to account for these types of inexact timeframes on the drift of a feature model. Feature model drift is a change to a feature model at a future point in time. We introduce a new concept, which we call the *change epoch*, which is the period of time during which a change due to feature model drift is in effect.

Each change epoch includes both a start time and a duration. For example, a supplier may phase out a component in 3-5 years, causing the feature model to have several modifications. Let, E_i be the change epoch of the i_{th} set of

changes that need to be applied to the feature model as a result of feature model drift. When the E_i change epoch is in effect, it means that its starting point is E_i^{start} and $3 \leq E_i^{start} \leq 5$. The duration of the epoch, E_i^{dur}, is $E_i^{dur} = \infty$.

To express feature model epochs, constraints must be added to bound the values for E_i^{start} and E_i^{dur}. We introduce the function,

$$S(E_i^{start}, E_i^{dur}, F_0, F_1, \ldots, F_{end})$$

to determine the begining of a change epoch as a value of time and the configurations of the feature model at each step. For example, if a supplier was expected to phase out a part 3-5 years in the future, then:

$$3 \geq S(E_i^{start}, E_i^{dur}, F_0, F_1, \ldots, F_{end}) \geq 5$$

Similarly, a separate function,

$$W(E_i^{dur}, E_i^{dur}, F_0, F_1, \ldots, F_{end})$$

calculates the duration of the change epoch. In the case of a part phased out of existence, the duration of the change epoch would be indefinite, or:

$$W(E_i^{dur}, E_i^{dur}, F_0, F_1, \ldots, F_{end}) = \infty$$

An important note is that this approach assumes that the changes that are applied to the feature model during a change epoch are assumed to be correct. For example, if a feature is removed in a particular step, any other modifications to the feature model needed to bring it to a valid state (*e.g.*, removing dependent cross-tree constraints, adding replacement features, etc.) are also applied so that the feature model does not have inconsistent or unsatisfiable constraints. Moreover, the approach also assumes that objective functions for the optimization process are not specified in a manner that they are undefined when one or more features are added or removed. At all steps, it is assumed that the objective function is defined and all features needed to calculate its value are present.

8.3.3 Epoch-based feature model constraints

The feature model drift epochs make it possible to model situations in which the exact step in which a change will occur to a feature model is not known. Instead, constraints are placed upon when the feature model drift epochs will occur and their duration. In order to account for epochs in the multi-step configuration CSP, additional constraints must be added. In the previous examples, if the j_{th} feature is an optional child of the i_{th} feature (the software package) at step T and at step K, the j_{th} feature becomes mandatory, we can model this as:

$$(f_{jT} = 1) \Rightarrow (f_{iT} = 1)$$

At Step K, the j_{th} feature becomes mandatory, changing the constraints on selection of the feature:

$$(f_{iK} = 1) \Rightarrow (f_{jK} = 1)$$
$$(f_{jK} = 1) \Rightarrow (f_{iK} = 1)$$

Now, assume that the j_{th} feature is an optional child of the i_{th} feature (the software package) at the start and at some step, K, where $3 \leq K \leq 5$, the j_{th} feature becomes mandatory, we can no longer directly model this as before. Instead, we must define the enforcement of the new feature model constraint in terms of its feature drift epoch. In this situation, we model this as:

$$(f_{jT} = 1) \Rightarrow (f_{iT} = 1)$$

If Step K is within the time period of the feature drift epoch, the j_{th} feature becomes mandatory, changing the constraints on selection of the feature:

$$((f_{iK} = 1) \Rightarrow (f_{jK} = 1)) \Leftrightarrow (E_i^{start} \leq K \leq E_i^{start} + E_i^{dur})$$
$$((f_{jK} = 1) \Rightarrow (f_{iK} = 1)) \Leftrightarrow (E_i^{start} \leq K \leq E_i^{start} + E_i^{dur})$$

where:

$$3 \leq E_i^{start} \leq 5$$

Using the concept of a feature model epoch, developers can encode ambiguity into the feature model drift. Developers can model periods of time during which changes are expected and reason about how variations in when those epochs occur will impact configuration. Most importantly, feature model epochs allow developers to create configuration scenarios that more closely mirror the uncertainty in real-world development at when a particular feature will be completed and become part of a feature model.

8.3.4 Ordered epochs

Another issue that developers face is that the development or deprecation of a feature from a feature model is dependent upon the development or deprecation of several other features. For example, developers may know that the next generation of a mobile phone platform is going to support connectors that can communicate with an aircraft media server. Within one year from the time that this new mobile phone platform is developed, they will be able to develop a video player that streams media from the aircraft media server on the same mobile platform.

In this scenario, the development of the mobile phone server video streamer feature is dependent upon the occurrence of the mobile platform's

server communication feature. The exact point in time at which the diagnostic interface feature will be developed is only known relative to the occurrence of another epoch. We term these types of epoch constraints, *ordered epochs*.

Using the modified model of multi-step configuration, we can defined an ordered epoch by constraining an epoch's start, E_j^{start}, and duration, E_j^{dur}, in terms of another epoch, E_i. For example, if we wish to define the epoch, E_j, as occuring at least two steps after the epoch, E_i, we can say:

$$E_j^{start} \geq E_i^{start} + 2$$

8.3.5 Feature drift branches

Using these CSP constraints, developers can encode ordering into the occurrence of epochs. Another key attribute of epoch ordering is the ability to introduce branching into the occurrence of epochs. For example, developers might have a single physical connector on a in-seat screen that they plan to use either to consume streaming video from a media server or connect to controls for managing the above seat lighting and vents, but not both.

To encode branching constraints into feature model drift, developers can use the E_i^{start} variable to encode branching constraints. For example, if the changes described by the i_{th} feature model drift are mutually exclusive with the changes in j_{th} feature model drift, this constraint can be encoded as:

$$E_i^{start} \geq 0 \Leftrightarrow E_j^{start} = -1$$
$$E_j^{start} \geq 0 \Leftrightarrow E_i^{start} = -1$$

where, $E_j^{start} = -1$ indicates that the j_{th} feature model drift never is in effect. Using this same strategy, arbitrary constraints on the branching of feature model drift can be encoded into the CSP.

8.4 Summary

Many SPL scenarios require developers to evolve a configuration over multiple steps, rather than in a single step. Multi-step SPL configuration, however, must take into account constraints on the change between successive configurations, such as the increase in cost of an aircraft's configuration from one year to the next. Moreover, even though configuration is performed over multiple steps, a valid configuration must still be produced at the end of each step (*e.g.*, prior to shipping the new year's model aircraft), which further complicates maintaining a functional system configuration.

It is hard to determine a sequence of feature model configurations and feature selections such that an initial configuration can be transformed into a desired target configuration. This chapter introduces a technique, called the *MUlti-step Software Configuration probLEm Solver* (MUSCLES), for modeling and solving multi-step configuration problems. MUSCLES represents the problem as a CSP, which enables CSP solvers to determine a path from a starting configuration to a target configuration. The output from MUSCLES is a valid sequence of feature selections that will lead from a starting configuration to the desired target configuration, while accounting for resource constraints.

Chapter 9

Supporting distributed product configuration

A las vitaminas se le van a oxidar las vitaminas,
pero ese es el precio del éxito.

Juan Manuel Tirado, Researcher

In industrial settings, products are rarely developed by one organization alone. Instead, software vendors and suppliers typically maintain their own product lines, which the contributes to a larger (multi) product line or software ecosystem. It is unrealistic to assume that all participating organizations agree on using a specific variability modeling technique—they will rather use different approaches and tools to manage the variability of their systems. We present an integrative approach that provides a unified perspective to users configuring products in multi product line environments, regardless of the different modeling methods and tools used internally. We also present a technical infrastructure and a prototypical implementation based on web services. We show the feasibility of the approach and its implementation by using it with three different variability modeling approaches (one feature-based, one OVM-style and one decision-oriented approach), i.e., the three most widespread types of approaches in the product line community. We show an example derived from industrial experience and present a performance validation of the tool-supported approach[†1].

[†1]Part of this material have been published in the SPLC conference[56] and the VAMOS workshop [55]

9.1 Introduction

Software product lines (SPL) are increasingly developed beyond the boundaries of a single organization [29]. For instance, in software ecosystems distributed organizations and teams create software products in a collaborative effort. Variability management and product configuration in such contexts need to reconcile the different modeling approaches, notations, and tools in use. Due to the significant differences in practices in different domains it is unrealistic to assume that there will ever be a single and standardized variability modeling approach despite ongoing standardization efforts[†2]. However, the increasing number of "island solutions" to variability modeling and product configuration restricts communication and hinders collaboration between distributed product line engineers. Especially in the context of software ecosystems [29], it is infeasible to assume one kind of modeling approach for all units of the ecosystem. The required coordination between the participating organizations (e.g., along the supply chain) further complicates this issue. Hence, there is a strong need for an integrative infrastructure enabling the collaboration between different organizations developing product lines. An approach is needed that supports different variability modeling languages, notations, and tools and that allows dealing with variability at different levels of granularity. For instance, variability might be modeled at the levels of features, architectural elements, or configuration decisions.

We propose Invar, an approach facilitating the integration of variability models[†3] created with different modeling approaches and potentially by different teams. In this chapter we focus on the product configuration aspects of our integrative infrastructure. Invar hides the internal technical aspects of using different variability models for configuration from the stakeholders performing the configuration. The specific tools or data formats (see [35, 103, 163]) used for defining the variability models are not relevant for the end users who only need to focus on the available configuration choices and their implications. Invar unifies configuration operations on variability models and allows modelers to freely choose a data representation by accessing variability models through web services. Our approach does *not* force organizations to *integrate* their configuration tools by adapting the internals of the tools. Instead, we allow them to *compose* their configuration mechanisms using wrappers and interface definitions. We validate our approach by integrating three different variability modeling dialects, i.e., feature modeling, orthogonal vari-

[†2]OMG Common Variability Language [170], http://www.omgwiki.org/variability/doku.php

[†3]Throughout this chapter, we use the term "variability model" to refer to product line models regardless of the specific approach and notation used, e.g., feature models, decision models, OVM models

ability modeling (OVM), and decision modeling. We also show how typical scenarios in software ecosystems can be supported with Invar.

9.2 The Invar Approach

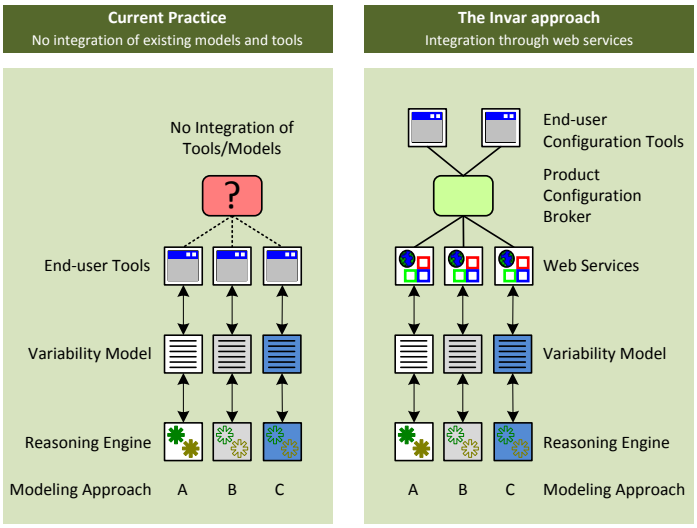


Figure 9.1: *A simplified view of model-based product configuration .*

Our Invar[†4]approach addresses these requirements by providing a "plug-and-play" approach to managing the integration of variability models. "Plugging" refers to simply adding new variability models to a shared repository. "Playing" refers to presenting the options provided by variability models to end users configuring a product. For this purpose, a variability model is treated as an autonomous entity, which can be plugged into the configuration space to provide configuration options. Autonomy however does not necessarily mean independence as variability models may be related with each other as shown in the ERP example. Our approach allows using variability models distributed across multiple repositories by accessing them through

[†4]Invar is a nickel-steel alloy notable for its uniquely low coefficient of thermal expansion. In metallurgy, it is a good example of how one can benefit from combining different metals, to achieve special desired properties. We chose the name Invar for our approach to reflect on the need to combine/integrate/compose different variability modeling approaches, notations and tools in the context of multi product lines. Alternatively, Invar stands for "**in**tegrated view on **var**iability".

web services providing configuration choices. An end user works with a front-end for product configuration and can use the services without knowing details about the concrete variability models made available by the services.

Based on the common characteristics of variability models described in Section §1, we defined a set of web services for accessing and using variability models. The operations and queries on the models, which are required for developing such web services, are based on our experience of developing product configuration applications [19, 21, 32, 54, 139, 140]. Figure §9.1 depicts the Invar approach compared with the current state of practice. In the current state of practice multiple heterogeneous variability modeling approaches are used by different organizations (left side). Different reasoning and analysis engines –for instance, SAT solvers [19] or rule engines [54] – are adopted for interpreting the models' semantics. There is no integration of the diverse tools supporting different notations. In practice organizations frequently define variability in configuration files or spreadsheets, which are typically not integrated with other variability modeling tools.

Using Invar, stakeholders create variability models using an approach of their choice (see right side of Figure §9.1). Invar defines key operations and queries (*configuration primitives*) on variability models to allow the integration of heterogeneous approaches. These configuration primitives are realized as operations of a Web Service to allow uniform access to the models. The participating organizational units can reuse variability models from other units even if they use different modeling approaches. Invar provides a single and transparent configuration tool to end users. This ensures interoperability and allows reusing variability models in different contexts. For example, one model may be shared between several companies and each one may use it to create different products. The participating organizational units could also create their own configuration tools and access diverse models via the Invar web services without having to know all the details about the actual modeling approaches underneath. It is even possible to integrate configuration files or spreadsheets into Invar by defining new web services that interpret these textual representations.

9.2.1 Configuration primitives

There are typical operations needed by end-user product configuration tools that "execute" or evaluate variability models. These operations are provided as methods by most of the existing APIs [21, 32, 54] allowing model-based product configuration. For example, the end user may query the set of all available options at a time, or send a request to select one of the options.

The basic concepts in most of these APIs revolve around *options* or *choices* for the model consumer. Typical operations on the models are:

Loading and initializing models: load() puts models from their persistent storage to memory, while reload() loads the model again from memory. The operation init() is used to start a new configuration session based on a model. save() persists the session for future use.

Querying the model for available options: For instance, nextQuestion() gets the next available question to be answered by the user, regardless if this is a feature, decision, or variation point. The approach only assumes that a question about an option or choice can be asked to a user. Analogously, previousQuestion() allows to obtain the previous question. peek() allows previewing the next available question without having to answer it. Walking through options step by step requires to define a certain order. For instance, for a tree-based structure this could be done breadth first or depth first. In our ERP example, such a walkthrough would start with a question about what key features the ERP solution should have (CRM, Project Management, Accounting) and could then continue with detailed questions on CRM, Project Management, or Accounting, depending in the answer to the first question and based on the related models. It could however also start with a question: "Do you need CRM support?", continue with detailed questions on CRM, and only later ask "Do you need Project Management support?".

Operating on the available options: The operation setValue() allows assigning a value to an option (e.g., it sets feature attributes or sets answers to decisions). For instance, setting the value (CRM;Project Management) on question "What key features shall your ERP solution have?" would constrain the following questions to CRM-related and Project Management-related questions while excluding Accounting (optional feature not selected in this case). The primitives select() and deselect() allow to explicitly decide about including or excluding a feature. This distinction is necessary as not selecting a feature is not the same as excluding it from the configuration. undo() and redo() allow canceling the last action or replaying the last selection. addOption() adds new options to available questions.

Notifications: The success() operation shows if an operation was carried out successfully while error() indicates problems. The primitive contradiction() shows whether the choices of the user are consistent with the model's semantics. The operations selected() and deselected() are used to inform users or other tools about user actions.

Obviously, this set of primitive operations is neither complete nor fixed. Depending on the end-user application there may be other operations that are

useful in building complex user interfaces. For instance, in some cases, several options can be proposed at the same time. Our approach thus can easily be extended to include new operations or queries on the models.

Keyword	Type	Description
isInit()	condition	This condition only evaluates to true when a user starts configuring a model for the first time. The actions connected with isInit() conditions are executed immediately after each model has been initialized on its Web Service.
isSelected()	condition	This condition evaluates to true whenever a specified option of a given question in a specific model is in the state selected. The condition is evaluated after each change of a user to a model, i.e., after the user answers a question related to the link.
isDeselected()	condition	This condition evaluates to true whenever a specified option of a given question in a specific model is in the state deselected. The condition is evaluated after each change of a user to a model, i.e., after a user answers a question.
doSelect()	action	Set a specified option, of a given question in a specific model to the state selected.
doDeSelect()	action	Set a specified option, of a given question in a specific model to the state deselected.
includeModel()	action	Add a model to the list of included models of the current navigation strategy, if it was not yet initially included or included by another action. Each model can only be included once.
addOption(...)	action	Add an option to one model as a child of an existing option. This is usually required, when a model extends another model.
inform(...), warn(...), recommend(...)	action	Display a specified message to the user, which can have the type *information*, *recommendation* or *warning*.

Table 9.1: *A summary of operations, which can be used to create IMDI links.*

9.2.2 Inter-model dependencies

Whenever a variability model is plugged into the Invar configuration environment, it needs to explicitly define its relationships to the other models. This is done by adding an *inter-model dependency information (IMDI) packet* together with the model. Dependencies between models are defined using

if *condition* then *action* clauses. These can be compared to conventional cross–tree constraints within one model. IMDI packets do not affect the internal semantics of the models in use. An IMDI action is executed when its condition evaluates to TRUE.

A summary of conditions and actions, currently supported by the Invar framework is listed in Table §9.1. In the following we describe the basic types:

Inter-model constraints: If selecting or deselecting an option in one model has implications on other models, an inter-model constraint needs to be defined. The conditions isSelected() and isDeselected() are used to specify such constraints. The corresponding actions are doSelect() and doDeSelect().

Informative actions: Modellers can also define actions such as inform(), recommend() or warn() that are intended to inform the users without changing the models.

Conditional inclusion of models: If several variability models are used for product configuration the order of presenting the models to the end user has to be defined. We define the action includeModel(), which changes the presentation order. This influences the model navigation strategy, i.e., which model is configured in which order.

9.2.3 Integrating variability models: the Invar architecture

The five main components of the architecture of our infrastructure are shown in Figure §9.2 (numbered in the figure).

(1) Vendor model repositories: Product vendors or suppliers add their variability models to model repositories. The models may or may not contribute to the same product and are not necessarily dependent on each other.

(2) Invar repository: The Invar repository defines aggregations of different models from the vendor repositories by logically grouping them. For instance, one particular model may be part of multiple product lines, as it may contribute to more than one product.

(3) Configuration web services: The different models residing in (possibly distributed) repositories are accessed by configuration web services. A Web Service provides a standard interface for different configuration front-ends such as websites, mobile devices, or stand-alone applications. For each type of model, designated configuration services are developed (by implementing an interface) that can read the data formats, interpret the content, and perform

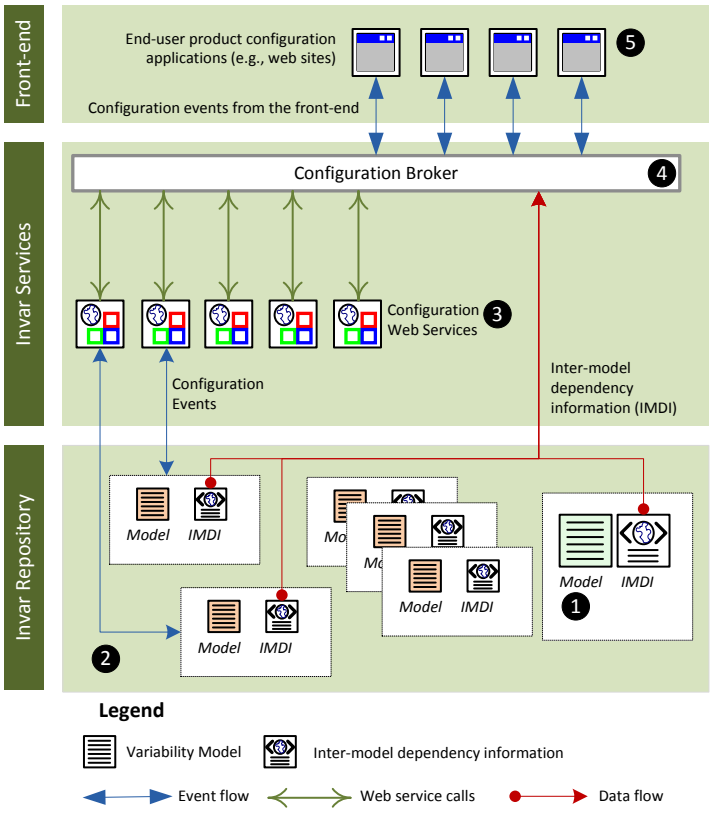


Figure 9.2: *Architecture of the* Invar *infrastructure. .*

operations on the models.

(4) Configuration broker: The configuration broker enables the communication between the web services. It reads the inter-model dependency information to determine which web services are affected when products are configured. The configuration broker also translates events from the end users and passes them on to the web services that need to react to the end user's interactions.

(5) End-user product configuration front-ends: The configuration choices defined in the variability models are presented to the end user in a product configuration front-end. This can be a website or a stand-alone application. We provide an example user-interface through the Invar framework website at http://invar.lero.ie.

9.2.4 Configuration service and enactment support

Central to Invar is the generic configuration interface defined for accessing the diverse variability models. The configuration service definition has to be implemented *once* for each modeling notation. The Web Service is designed such that the configuration options are presented to the end user as questions. Questions are only a means to render the variation point to present it to the user. This means the user is asked questions about a certain "feature" (in the wider sense) or a property of the system she configures. The possible answers to the question (the available alternatives) are presented to the user such that she may choose one or many of them depending on the type of variability. The notion of "questions" and possible answers as options is therefore key to the Invar configuration service. The interface consists of two parts: the *variability model query part* provides basic information about models (e.g., the set of available questions and the possible answers). The *operational part* directly interacts with the models to assign answers to specific questions (e.g., when selecting a particular feature). The configuration service also defines a set of predefined question types. The types have been defined based on how the end user is supposed to answer them. For example, the question type Alternative refers to questions where the user can select exactly one option (rendered using radio buttons or comboboxes in the UI); for Optional the user can pick multiple items (rendered using checkboxes in the UI) and MoreThanOne refers to cardinality (1..*) (rendered using multi-selection checkboxes in the UI).

9.2.4.1 Enactment strategies

When configuring a model in Invar the user is prompted to answer questions comprising a set of options a user has to select or deselect. The order in which such questions are presented to the user can vary depending on different criteria. Invar provides two mechanisms to help product line engineers define in which order the questions need to be presented to the user.

The first mechanism uses *predefined orders* to present configuration options to the user. Three orders depending on the way a tree can be traversed, i.e., *in-order*, *pre-order* and *post-order*, are available for feature models. Invar also offers two ordering styles based on the number of options the questions have *"more to less"* and *"less to more"* available for feature models and orthogonal variability models. For decision-oriented models, the order of questions is either given through their visibility conditions [54] or can be defined similar to the definition used for feature models. Finally, Invar also supports an alphabetical order and a random order for all types of models. More details

about the offered orders in each plugin are presented in Section §13.

Alternatively, invar provides a second mechanism, i.e., an interface, by which the engineer can define a *custom order*. This interface defines two methods, i.e., getOrders() that returns the names of the orders and getQuestions(String order) that returns the list of sorted questions given a particular order.

The order in which questions are answered can be important from an end-user point of view but also can have an impact on the performance of the back-end configuration tools [111]. We present experiments to evaluate Invar's performance in Section §13.3.

9.3 Summary

In contrast to software product lines, software ecosystems go beyond organizational boundaries per definition. To operate in such contexts, it is equally important to integrate data, models, and tools across organizations and inner-organizational boundaries; together with the alignment of the business strategy among the cooperating units. From the perspective of software reuse and configuration, the initiation and growth of a software ecosystem can be fostered considerably if the involved stakeholders are supported with tools and techniques for dealing with variability in a uniform manner.

In this chapter, we presented an approach to facilitate the integration of variability models during product configuration regardless of the modeling techniques, notations, and tools used. Based on an illustrative example, we defined the basic user interactions required for configuration in general (configuration primitives) and mapped these interactions to the concrete semantics of individual modeling approaches, i.e., an approach for feature modeling, and OVM-based approach, and an approach for decision modeling. We provide a technical infrastructure and a prototypic implementation of an integrative approach based on web services that supports defining dependencies between multiple variability models as well as different strategies for model enactment during product configuration. We showed the feasibility of the approach and its implementation by applying it to three different variability modeling approaches and by showing that we are able to integrate them in the context of an example ERP system. We also provided initial evidence on the performance of the approach during configuration. One of the challenges to achieve industrial-scale application of this approach is the effort required for modeling the configuration projects in Invar. Each model has to explicitly define its

relationships to the other models and, if options in one model have implications for other models, those inter-model constraints (dependency links) need to be defined.

Part V

Validation

Chapter 10

Optimizing the Android emulation in the cloud

Amigo de muchos, amigo de ninguno.

Dicho popular, Andalusian people

Developers for Google's Android smartphone platform are increasingly facing the challenge of fragmentation, whereby a growing number of distinct Android hardware/software configurations are forcing developers to test their application much more extensively than desired. For example, a report by OpenSignalMaps found that their application was installed on 4000 unique device configurations, which would require significant testing time and effort to completely cover. Due to the large number of in-use platform configurations, it is difficult for developers, regardless of development team size or proficiency, to test their software products on all or even most platform configurations before release. A balance of comprehensive testing and reasonable consumption of testing resources (e.g. money, time, computation power, storage, etc) is needed[†1].

[†1]This chapter is based in [71] and part of this material is in press in the SQJ journal

In this chapter, we use TESALIA to enable the testing of Android applications in the cloud while optimizing the market-share coverage of the app and reducing the cost of the execution. Figure §10.1 shows the different steps TESALIA takes to optimize test selection, including examining product variability, product computation cost, and product market share to identify valuable testing configurations. In order to perform this analysis, we have represented the configuration of the Android platform using a software product line, with the Android emulator being one possible product feature. By using a software product line to represent many different mobile platform features of an Android device, such as such as screen resolution or communication capabilities (*e.g.*, 3G, LTE, etc.), we were able to easily apply TESALIA and optimize the test selection. Using the current Android emulator, which contains 46 features, we can test up to 2^{46} unique platform configurations. Note that, we conservatively calculate the maximum number of unique platform configurations by using binary options and mutual exclusion.

Figure §10.1 shows the process we performed to validate our approach. First, we describe how we extracted value and cost information (market share and cloud usage) and how we used it to attribute a feature model describing the Android emulator options. Later we go through the four different experiments we developed to test the prioritization, pruning and packaging capabilities of TESALIA.

One of the most common approaches is to pick a set of the most popular mobile devices and then to test exclusively for that subset of devices. For example, the Android Skype application is installed on thousands of unique Android platform configurations [†2], but the Skype application is only officially certified to work on fewer than 25 Android devices [†3].

10.1 Experimentation data

In order to execute the analysis operations presented in Section §6.3 a feature model with cost and value information needs to be built. To build this model, we used the Android emulator as the basis of our analysis. First, we encoded the variability present in the emulator options. Second we extracted cost value from the costs of executing a test in a cloud. Finally, we obtained publicly available market share data to create value attributes [†4]. In the rest of this section we provide detailed information about how we built the inputs

[†2] http://opensignal.com/reports/fragmentation.php
[†3] http://www.skype.com/intl/en-us/get-skype/on-your-mobile/skype-mobile/android/
[†4] http://developer.android.com/tools/help/emulator.html

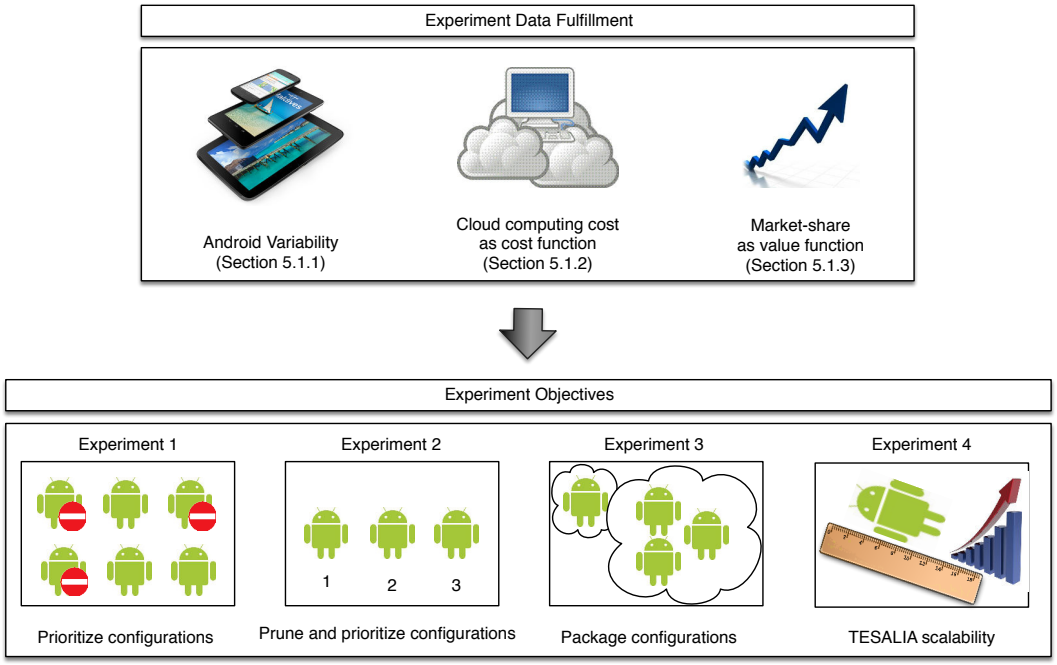


Figure 10.1: *Steps for optimizing the testing of Android apps using TESALIA.*

for the three experiments shown in Section §10.2.

10.1.0.2 Representing all valid Android testing configurations using a feature model

Feature models describe individual features, or units of functionality, using a tree-like structure, as shown in Figure §10.2.In the model presented in Figure §10.2 abstract features have been used to group the characteristics of the different mobile platform configurations defined in the Android ecosystem.

Using a feature model of a mobile software platform's configuration rules, the goal is to select platform feature configurations that adhere to the constraints encoded into the feature model. An important property of feature models is that there are well-known approaches for automatically deriving valid or optimized feature configurations from a feature model by converting the model to a constraint satisfaction problem (CSP) or a SAT problem.

TESALIA uses the CSP formulations presented in Section §6.3 to automat-

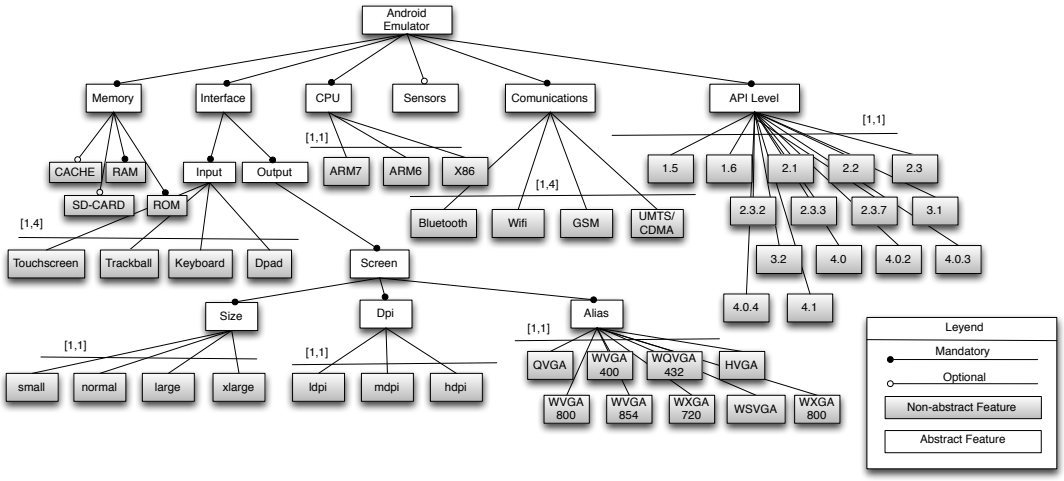


Figure 10.2: *The Android feature model.*

ically prune, prioritize, and package the set of platforms configurations to test. Figure §6.3 shows the general process carried out by TESALIA. Moreover, users can scope the testing process by annotating features that have to be present or not in the products. Also, users may want to add 'complex' constraints representing any other exclusion between mobile platform features or costs related to the model, such as *"If the mobile platform configuration contains Android 1.5, then the mobile platform configuration should not contain more than 250mb of RAM memory."* Note that those annotations are not present in the feature diagram presented in Figure §10.2. They are inputs for the different operations and may vary depending on user preferences. The feature model representation allows us to exclude invalid platform configurations (a.k.a software product line products) and to dynamically construct a CSP for pruning, prioritizing, and packaging the products to test. In order to provide input for the operations presented in Section §6.3, we created a feature model representing the various options available to the Android emulator by extracting the rules outline in the emulator documentation. We included cross-tree constraints to ensure that no incorrect configurations are generated. As mentioned in Section §6.3.1, there are automated mechanisms that enable the extraction of feature models from a set of valid product configurations. However, when building the Android feature model, we found it simpler to identify constraints instead of listing valid configurations and therefore chose to directly create the feature model.

10.1.0.3 Using feature count as a proxy for the cost of execution in a cloud environment.

Cloud computing, which is promoted as a low-cost, effectively infinite computing solution, enables the average developer to temporarily rent computing resources powerful enough to test hundreds of unique platform configurations. Running such a test suite was previously impractical due to the prohibitive cost of assembling and maintaing computing equipment dedicated for testing. Commoditized cloud computing allows developers to perform large-scale application testing on multiple platforms with a minimal amount of added development time and zero equipment investment overhead.

Renting multiple virtual machines in the cloud can be expensive. To define the cost of testing a software product in a cloud, a tester must consider items such as the time required to test each configuration, the number of configurations under test, and the number of cloud virtual machines that must be rented. In Android, many platform features have complex non-linear interactions with other platform features, which affect both product performance and the resources needed to execute and test the Android emulator product. Modeling the exact computation time required to execute a single Android emulator test is beyond the scope of this work. Additionally, modeling the cloud resource usage of a subset of configurations, such as all configurations that use Android 1.6, by manually calculating the value of a few fully specific configurations. For example, {Android 1.6, screen size 240x320, 16MB RAM, has 'volume' hardware buttons, does not have 'Home' hardware button, etc} is difficult and also beyond the scope of this work.

To define the cost of executing an emulator instance in a virtual machine, we count the number of features included in each product. This means that instances executing more options (e.g. instances which are likely to consume more resources), have more impact in the calculated cost of a product. The $\sum$ function is used to calculate the cost of each product by making the total cost of a product equal to the number of concrete features composing that product.

10.1.0.4 Market-share as a value function

A key aspect of TESALIA is that it can rely on different value-functions, such as market-share data $f(VAC)$, for each individual mobile platform feature configuration. Although the process for obtaining this data is beyond the scope of this chapter, two potential approaches are briefly outlined. First,

a number of commercial vendors provide access to fine-grained mobile platform market share data, such as comscore [†5]. A second, more commonly used approach by application developers, is to directly instrument their application and collect platform configuration data. For example, Skype keeps track of all known platform configurations that their software has been executed on. In general, application developers may find it most effective to use data that they have obtained from instrumenting their application, since it provides an accurate picture of the market-share of platform features in-use by the consumers of the developer's application rather than the market in general. Figure §10.3 shows an example of the market share data that was obtained from Google [†6] and used for the experiments in this chapter.

An important attribute of the market-share driven approach to selecting mobile platform configurations for testing is that this approach can optimize the selection of platform configurations *with respect to the market-share coverage of the features that the application actually uses*. For example, if the five best selling Android devices are chosen for testing, these devices may be newer and have similar features to one another, such as large screens, that are not characteristic of the large installed base of previous generation phones. If screen size is a feature that has a direct impact on the application being tested, limiting the diversity of the screen sizes tested is not beneficial. Moreover, the app being tested will have a high likelihood of being installed on an older model device with a smaller screen size that it has not been tested on.

Because the mobile platform feature model can be pruned to focus on only the features that directly impact the application, it is possible to derive mobile platform configurations to test on that maximize the market-share coverage with respect to the subset of features that matter to the application. For example, a developer can choose to only support newer devices with Android version 4.0 and higher by pre-labeling the corresponding feature variable in F with value 1 (*e.g.*, requiring every derived mobile platform configuration to include it). In this case, the developer can maximize the market-share coverage with respect to the devices that actually have the supported platform version.

The market-share data used in the experiments was freely obtained from Google Android market as a whole rather than for a specific application. We focused on a subset of the market-share data related to screen size, resolution, and Android version, since these are mobile platform features that impact the vast majority of Android apps. For other platform features, the experiments used a uniformly distributed market-share valuation. For

[†5] http://www.comscore.com

[†6] http://developer.android.com/about/dashboards/index.html/

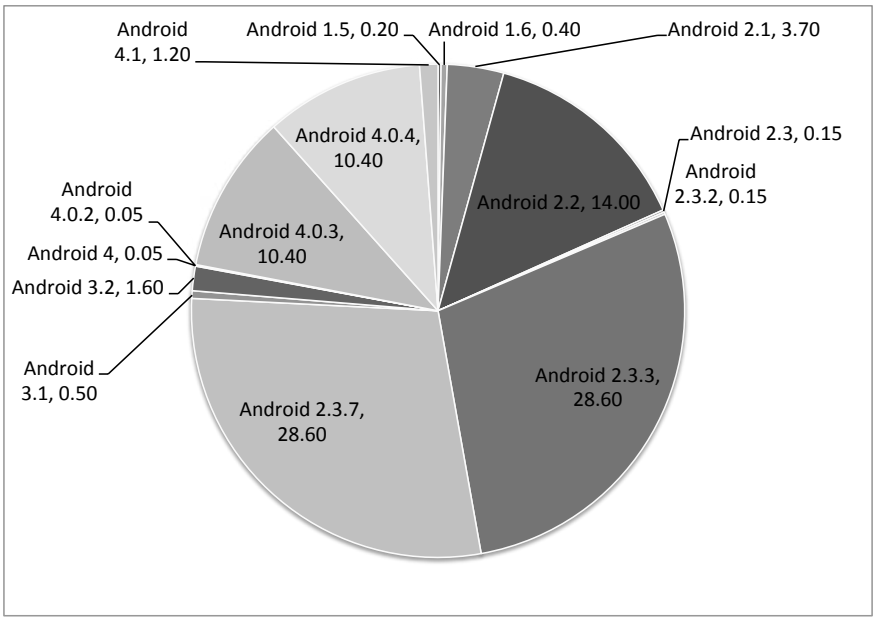


Figure 10.3: *Market-share example data by September'12.*

example, the market-share data of the communication options is fixed to $1/numberOfAvailableOptions$, and with four options (GSM, CDMA, Bluetooh and Wifi) every option will be fixed to 25% market share. TESALIA generally performs better in a real-world non-uniform scenario, as optimized mobile platform selections have a substantially greater market-share than non-optimized choices in this type of model. We chose to use the uniform model because we had incomplete data on some platform features.

10.2 Experiments

Four different experiments were conducted to evaluate the TESALIA approach. These experiments compare and analyze: 1) the market-share prioritization of TESALIA-derived mobile platform configurations for testing versus the market-share coverage of a set of the most popular phones on Amazon; 2) a cost/benefit analysis of the configurations produced by TESALIA versus selecting a set of popular mobile phones using the pruning and prioritization operations; 3) the TESALIA validity for packaging configurations using the Android feature model attributed with Google market share; and 4)the TESALIA scalability with different model sizes and topologies.

Hypotheses of Experiment 1	
Null Hypothesis (H_0)	TESALIA does not optimize the percentage of market share covered features versus the traditional approach of buying the most sold phones.
Alt. Hypothesis (H_1)	TESALIA optimizes the percentage of market share covered features versus the traditional approach of buying the most sold phones.
Dependent variable	The market share coverage.
Blocking variables	The most sold phones and the market share indexes.
Model used as input	*Android feature model presented in Figure §10.2*
Hypotheses of Experiment 2	
Null Hypothesis (H_0)	The use of TESALIA will not result in a higher market-share impact metric than selecting the most commonly sold phones, for a given maximum budget.
Alt. Hypothesis (H_1)	The use of TESALIA will result in a higher market-share impact metric than selecting the most commonly sold phones, for a given maximum budget.
Model used as input	*Android feature model presented in Figure §10.2*
Blocking variables	The most sold phones, market share indexes and the maximum cost allowed set to 600$.
Model used as input	*Android feature model presented in Figure §10.2*
Hypotheses of Experiment 3	
Null Hypothesis (H_0)	TESALIA cannot derive the products to test in a reasonable time.
Alt. Hypothesis (H_1)	TESALIA is able to derive the products that maximize the value function in less than 5 minutes.
Model used as input	*Android feature model presented in Figure §10.2*
Hypotheses of Experiment 4	
Null Hypothesis (H_0)	TESALIA cannot cope with models having 2000 hundreds features.
Alt. Hypothesis (H_1)	TESALIA scales up to models having 2000 features.
Blocking variables	Number of features: 10,20,30,40,50,100,200,300,500,1000,2000 Percentages of cross-tree constraints: 5%, 10%, 15% Number of complex constraints: 1,2,5
Constants	
CSP solver	*ChocoSolver v2*
Heuristic for variable selection in the CSP solver	*Default*

Table 10.1: *Hypotheses and design of experiments to evaluate TESALIA.*

Table §10.1 shows the hypothesis of the experiments executed to validate our approach. To make the experiments reproducible, a number of fixed assumptions are made, such as homogeneous feature costs. ChocoSolver [†7], with it's default heuristic, is used as the CSP solver for extracting software products from the feature model presented in Figure §10.2

Experimental platform - Virginia Tech ATAACK Cloud. The experiments were conducted using a version of TESALIA implemented in Java. Further, TESALIA was installed in the Virginia Tech ATAACK Cloud, which is a cluster testbed capable of simultaneously testing many configurations of an Android application on 1,000+ Android Emulator instances. The ATAACK Cloud runs on 34 dual-CPU Dell Blades with Intel Xeon X3470 CPUs running at 2.93GHz, with 16 threads per CPU, and CentOS v6. Each dual-CPU Dell Blade has 36GB of RAM.

10.2.1 Market-share based prioritization.

In order to analyze TESALIA's market-share optimization capabilities, we designed an experiment to compare the market share of TESALIA-derived configurations with that of the 20 phones with the highest sales volume on Amazon. Our hypothesis is that simply selecting a set of the most-sold phones will not provide the best market-share coverage and that TESALIA's solutions will provide more market-share coverage. For this experiment, we obtained market share data from Amazon [†8]. For the mobile platform configurations, we look at the 20 first configurations recommended by TESALIA using screen size and density mobile platform features. Note that only screen market share data have been used for this experiment.

In the Table §10.2 we present the raw data for the twenty most sold phones. This table summarizes the screen size, the pixel density, the market share categories and the market-share coverage. Note that only the prioritization operation has been performed in this experiment. Later in Table §10.3 we show the market-share data provided by google in October'12, which is the most recent market-share data that is freely available.

The market share coverage refers to the kinds of screen (combinations of dpi and size) covered by a concrete product and its associated market share value. Thus, having one configuration with a small and ldpi configuration will provide the 1.7% market-share coverage.

[†7] http://www.emn.fr/z-info/choco-solver/

[†8] Amazon provides the list of the top-selling phones in the United States as of December 12 (www.amazon.com)

Phone Name	Screen size	ddi	dpi category	Screen size category	MS coverage
Samsung Galaxy S III	1280x720	306	XHDPI	LARGE	3,6
Samsung Galaxy Note II	1280x720	267	XHDPI	LARGE	3,6
Samsung Galaxy S5830 Galaxy Ace	480x320	164	MDPI	NORMAL	11
HTC Droid incredible	800x480	252	XHDPI	NORMAL	25,1
Samsung Y Galaxy S-5360	320x240	133	LDPI	SMALL	1,7
Samsung Galaxy Nexus	1280x720	316	XHDPI	LARGE	3,6
Samsung Galaxy SIII mini	800x480	233	HDPI	NORMAL	50,1
Samsung Galaxy sII	800x480	219	HDPI	NORMAL	50,1
HTC A9192 inspire	800x480	217	HDPI	NORMAL	50,1
Motorola Atrix mb860	960x540	275	XHDPI	NORMAL	25,1
Sony xperia U ST25A-BW	854x480	280	XHDPI	NORMAL	25,1
Dell Aero	640x360	210	HDPI	NORMAL	50,1
HTC EVO 4g	1280x720	312	XHDPI	NORMAL	25,1
Samsung Galaxy Y Duos	240x320	127	LDPI	SMALL	1,7
Samsung galaxy gt-s7500 ACE plus	480x320	158	MDPI	NORMAL	11
Google nexus 4	1280x720	318	XHDPI	LARGE	3,6
Samsung Galaxy ace 2	480x800	246	HDPI	NORMAL	50,1
Sony xperia play	480x854	245	HDPI	NORMAL	50,1
HTC freestyle f5151	480x320	180	MDPI	NORMAL	11
Motorola droid 2	480x854	265	XHDPI	NORMAL	25,1

Table 10.2: *Twenty Amazon most sold phones (December'12).*

	ldpi	mdpi	hdpi	xhdpi
small	1.7%		1.0%	
normal	0.4%	11%	50.1%	25.1%
large	0.1%	2.4%		3.6%
xlarge		4.6%		

Table 10.3: *Google provided market-share (October'12).*

Results & analysis→ Figure §10.4 shows the market-share coverage of the first 20 most sold phones with respect to screen size and density compared with the market-share covered when buying the most sold phones in Ama-

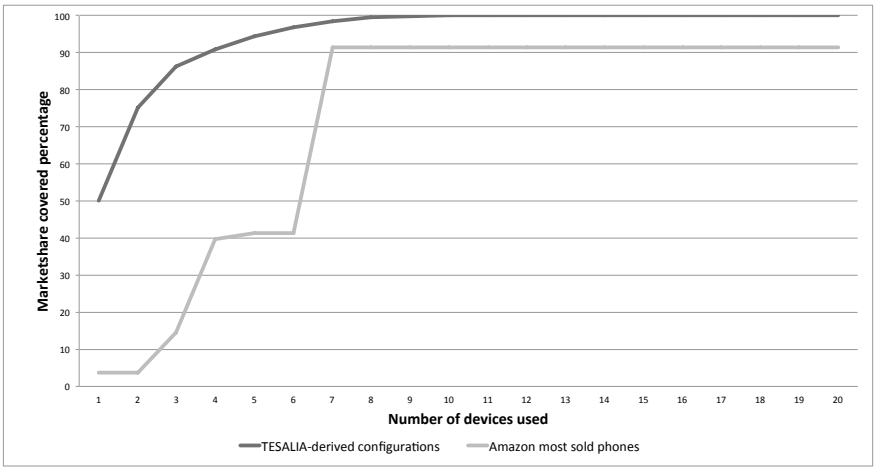


Figure 10.4: *Market-share coverage: Amazon vs TESALIA.*

zon. We can see in the chart of Figure §10.4 that is not possible to reach the 100% market share coverage when relying in buying most-sold phones. Also, using TESALIA-derived configurations we reach the 90% of market-share coverage with only four configurations. Moreover, if we only take into account the five first phones we observe that the five TESALIA-derived configurations cover over 53% more of the market than the five most sold phones. The overall market-share covered buying the five most sold devices is 51.4% while the five most-scored TESALIA-derived configurations cover 94.4% of the market. In light of these results we can conclude that the benefits of using market-share prioritization enable developers to test their applications in front of those devices their customers use. Moreover, we observed that buying the flag-phones or most-sold phones is not a good idea when optimizing the testing of Android applications.

10.2.2 Cloud cost and market-share optimization

This experiment compares the cost per unit of market-share obtained from TESALIA-derived configurations versus the most sold phone sets from Experiment §10.2.1. Each phone was considered to have a cost of $600 in this experiment, which is typical of unlocked mobile devices. Then we calculated the cost of buying the N first phones and divided by the market-share obtained using them. This is,

$$\frac{600 \times \mathrm{NumberOfConfigurations}}{\mathrm{marketShareCovered}}$$

For example, for the first Amazon phone we would consider the cost as $^{600}/_{3.6}$, for the third $^{600\times3}/_{14.6}$ and so on. With this experiment we want to show the economical benefit of automatically derive the most valuable configurations in terms of market-share.

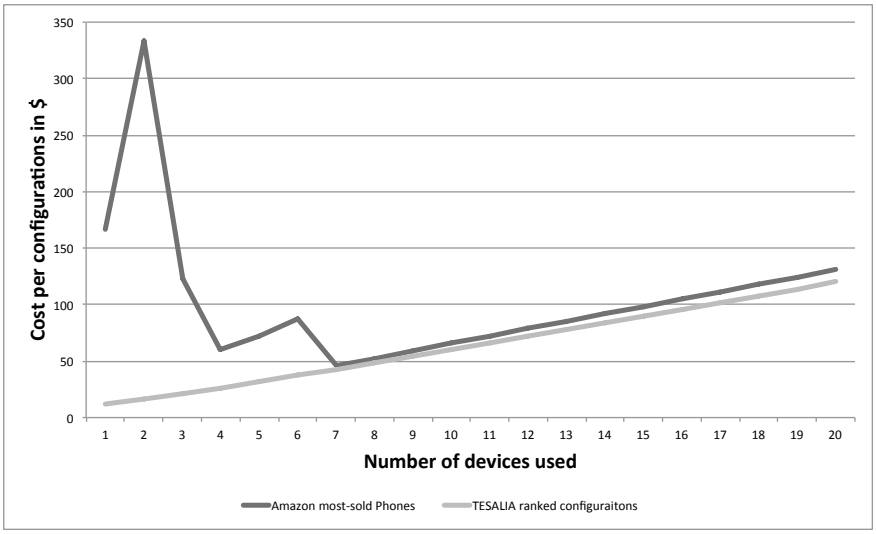


Figure 10.5: *Market-share coverage per monetary unit.*

Experiment results. Figure §10.5 shows the market-share/cost when using TESALIA versus buying the N most sold phones. As can be seen in the experiments, TESALIA is more cost-effective than selecting a set of the most popular phones to test on. Moreover, another interesting conclusion is that testing more than 10 of the most sold phones or TESALIA-derived configurations provides no additional benefit. The surplus saved when using the TESALIA approach could be invested either into additional testing on other mobile platform configurations or running extra tests on the same mobile platform configurations.

10.2.3 TESALIA-packaging validity for the Android scenario.

A key question when using CSPs and solving NP-hard problems, such as Knapsack problems, is the scalability of the approach. The purpose of this experiment is to determine if TESALIA is scalable enough to solve market-share optimization problems for real-world mobile platform configuration spaces and market sizes. This experiment analyzes the time for TESALIA to derive all possible configurations of the Android platform that represent real devices

and solve the Knapsack problem to optimally select a set of market-share maximizing configurations to test.

Experimental platform → This experiment was conducted on an Intel Core i7 processor running at 1.8GHz, with 4GB DDR3 RAM and the 6.0 Java Virtual Machine with command line options –Xmx250m and –Xms64m. The operating system was Mac OS v.10.7.1.

Results → Our experiments showed that TESALIA can find 29,510 configurations per second. Based on estimates of the total number of Android devices from OpenSignalMaps, we estimate that there are roughly 4,000-6,000 Android device models. Moreover, we attempted to overestimate the number of possible Android versions that could be on each device as 5, since newer devices probably have fewer versions of Android that can run on them and we consider 5 to be a reasonable upper bound on the number of potential different Android versions available for a given device model. Given this estimate, there could be up to 30,000 different valid platform configurations in the market, all of which could be derived by TESALIA in approximately 1 second.

We also tested the time to derive package the set of tests with 30,000 products. Solving this problem on our testbed took an average of 232 seconds on our test platform. We solved the Knapsack using an exact technique. Substantially faster approximation algorithms are available that give near-optimal results in substantially less time. These algorithms could also be used if desired.

10.2.4 TESALIA scalability.

The scalability of TESALIA has been validated for its use in current Android feature model. However, is important to verify that it scales for larger problems. We tested TESALIA in front of two different models sets. First, random models and later, models from the SPLOT [112] repository.

Testing TESALIA with random models → The random feature models used vary between 10 and 2000 features, 0 to 15 % of cross-tree constraints and 0 to 5 complex constraints (A constraint involving attributes). To generate these models, we used the Betty tool [156] provides an implementation of an algorithm [174] for the feature model random generation. Later, we configured it to generate models having two integers attributes per feature (value and cost) in a range from 0 to 10. Also, to prevent possible threats affecting the experiments execution, we obtained 10 models per combination of inputs and present the average time. Moreover, non-valid models were discarded for this experiment. Finally, three different functions were defined for each test-

ing scope. For pruning, we discarded the configurations exceeding a total cost greater than the 10 multiplied by the number of features. For prioritization, the function used is the sum of the value attribute. Finally, for the packaging scope, the total cost allowed in the knapsack problem is twice as the maximum cost for a product.

Testing TESALIA with SPLOT models → While the random models allow us to see the impact of the different variables in the execution time, they may not mimic the topology of realistic models such as those used in video-sequence generation domain[69] or cloud computing usage[73]. To determine if TESALIA scales in a realistic scenario we took all SPLOT models[112] and randomly added a cost and a value attribute per feature so that we could execute our optimization. We also added 0 to 5 complex cross-tree constraints per model. Attribute values ranged from 0 to 10 in the random models. Also, we generated 10 attributed models per combination of inputs and presented the average time. Again, we used the three same objective functions used for the random models.

Experimental platform → This experiment was conducted on of the blades of the ATAACK cloud in one thread. Concretely, the blade was using a dual-CPU Dell Intel Xeon CPU X3470 running at 2.93GHz CentOS v6 and the 6.0 Java Virtual Machine with command line options –Xmx250m and –Xms64m.

Results with random models → Our experiments show that TESALIA scales up to models with 2000 features without requiring more than 20 seconds of CSP computation for any of the analyses. Figure §10.6 shows the average time required in a logarithmic scale by each permutation of input variables along with maximum and minimum values (marked by points above the bars). In the worst case, the execution of the operations did not exceed 10 seconds, which we consider is sufficient for a wide range of applications. Also, in the figure, we can observe that the time required by TESALIA depends on the testing operation being used. That is, different operations perform differently. For example, summing the execution time for all input parameters combinations and pruning the tests takes 7290.5 milliseconds less than test prioritization.

Results with SPLOT models → After executing the three operations with the models available in the SPLOT repository we noticed that 7.421 milliseconds were required for the pruning operation, 8.446 for the prioritization, and 11.099 for the packaging operation. Figure §10.7 shows a small set of the models analyzed, which contains the most CPU consuming models, and 0 complex cross-tree constraints. For example, the most complex model in the repository – REAL_ FM_4 – took 524.5 milliseconds for the packaging operation. With these results in hand, we can conclude that our approach scales up to real-

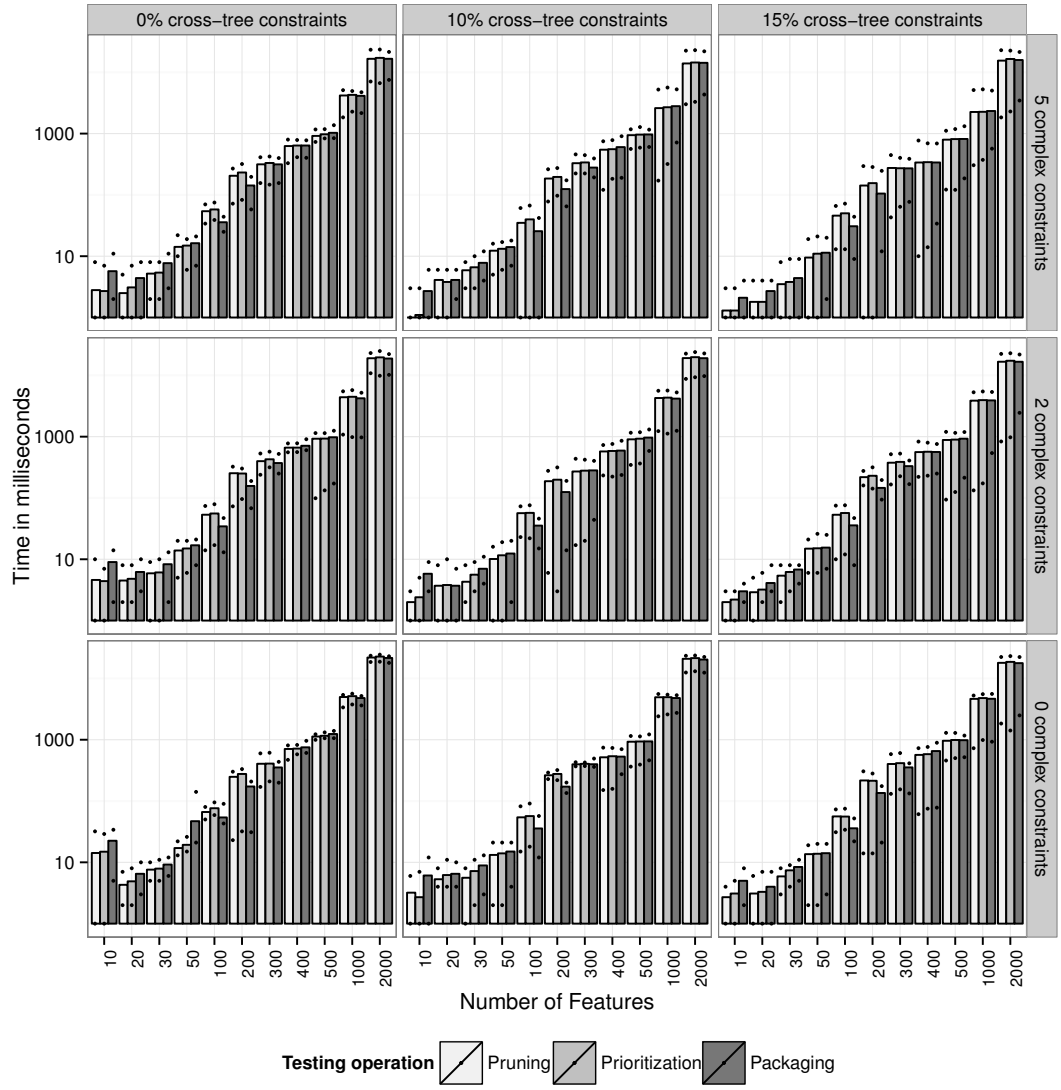


Figure 10.6: *Time required by TESALIA's operations with random models.*

istic feature model sizes, as well as to large random models. Moreover, we can infer that the real models are less complex for the operations presented in this chapter. The data set with the time required for all SPLOT models can be found in the material website. We also noticed that the inclusion of complex cross-tree constraints reduces solving time as there are less configurations to explore.

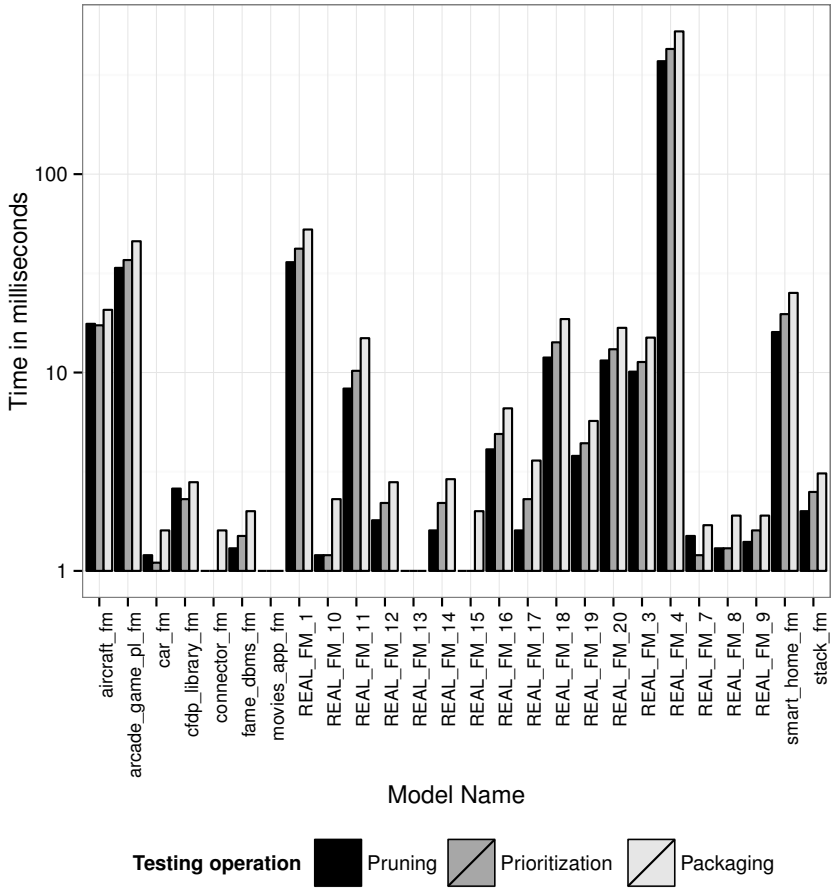


Figure 10.7: *Time required by TESALIA's operations with SPLOT models .*

10.3 Analysis and discussion

In this section we discuss the results we obtained and how the null hypothesis has been rejected, thus accepting the alternative hypothesis.

We performed four different experiments to check the validity of the TESALIA approach. With the first experiment we aimed to check the validity of the market-share metric when selecting smart-phones configurations to test. The null hypothesis was that TESALIA does not improve the market share coverage compared to testing on the highest selling phone configurations. To refute the null hypothesis we compared the 20 most sold phone configurations

from Amazon with the configurations produced by using TESALIA. Taking a look to the results we accepted the alternative hypothesis which indicates that TESALIA outperforms the traditional approach of testing on the most sold phones.

In the second experiment we compared the cost of the traditional approach of testing on the most sold phones versus the TESALIA approach. We compared the cost per configuration in dollars assuming that each phone costs 600$. In this case, the null hypothesis to refute is that it is cheaper to buy the most sold phones than using TESALIA. In this case, again for the 20 most sold phones, the experiments did not refute the null hypothesis leading us to accept the alternative hypothesis: TESALIA is more cost effective than the traditional approach.

The third and fourth experiments investigated the scalability, in terms of computing cost, of our approach. In each experiment, we determined the execution time of the approach when provided feature models of varying complexity and structure. In both experiments, we have been able to refute the null hypothesis and conclude that TESALIA can properly cope with realistic models described in prior literature. We noticed that the number of constraints is not impacting that much the analysis process. However, the less constraints the model has, a small increment in computing time is observed, specially in the minimum and maximum values. We suspect that this is caused because we are maintaining the same value and cost functions, thus, the less constraints the model has implies more products, making more costly to traverse all solutions.

In these experiments we have shown that TESALIA can cope with the constraints and variability complexity present in Android and in a variety of other models. Moreover, we showed the benefits TESALIA can offer. However, we need to check the viability of the TESALIA approach with industrial partners in future work. In current work, we have begun developing guides for developers to aid in creating optimization functions [7] for a variety of domains.

10.4 Threats to validity

Even though the experiments presented in this chapter provide evidence that the solution proposed is valid, there are some assumptions that we made that may affect their validity. In this section, we discuss the different threats to validity that affect the evaluation.

External validity. The inputs used for the experiments presented in this

chapter were either realistic or designed to mimic realistic feature models. The Android feature model is realistic since numerous experts were involved in the design. However, since it was developed using a manual design process, it may have errors and not encode all device configurations. Also, the random feature models may not accurately reflect the structure of real feature models used in industry. Another possible threat is the lack of market-share data affecting more features and the age of the market share data. The major threats to the external validity are:

- *Population validity*, the Android feature model that we used may not represent all valid Android configurations. Also, random models might not have the same structure as real models (e.g. mathematical operators used in the complex constraints), also the use of incomplete market-share data may have introduced errors. To reduce these threats to validity, we generated the models using previously published techniques [174] and using existing implementations of these techniques in Betty [156]. Also, for the attributes generation, as there are no well-known values in the literature yet, we introduced extended cross-tree constraints and attributes ranges consistent with past studies of feature models [46] and [128]
- *Ecological validity*: While external validity, in general, is focused on the generalization of the results to other contexts (e.g. using other models), the ecological validity if focused on possible errors in the experiment materials and tools used. To prevent ecological validity threats, such as third party threads running in the virtual machines and impacting performance, the TESALIA analyses were executed 10 times and then averaged.

Internal validity The CPU resources required to analyze a feature model depend on the number of features and percentage of cross-tree constraints. However, there may be other variables that affect performance, such as the nature of the complex constraints used. To minimize these other possible effects, we introduced a variety of mathematical operators over the integers in the constraints to ensure that we covered a large part of the constraint space.

10.5 Summary

The large amount of diverse and variant mobile platform configurations existing in the Android ecosystem makes the testing of Android apps a costly

and error prone process. However, the use of cloud computing resources makes possible to use virtually infinite computing resources. Nevertheless, cost and time constraints narrow the amount of resources we can use for testing. Therefore, techniques were required to maximize and optimize thoseconflicting constraints.

In this chapter we have applied the TESALIA solution to this context and have effectively optimize the market-share coverage of the target platforms where we execute our tests. Moreover, we have succesfully stressed the scalability of our tool within the set of models from the SPLOT feature model repository.

For reviewing purposes, TESALIA and associated test data is available from http://tesalia.github.io/. Please note, that the software is distributed under an LGPLv3 license.

Chapter 11

Managing video-sequences variability

Every real story is a never ending story.

Michael Ende, Writer

Image recognition systems rely on signals captured by video cameras, which are then processed through a chain of algorithms. Basic signal processing algorithms are assembled in different ways, depending on the goal of image recognition (scene interpretation, follow a specific object, etc.). Each algorithm is a complex piece of software that can be configured through multiple parameters. Additionally, these algorithms can be assembled in different chains corresponding to various recognition needs, and processing very different kind of images (day/night, lots/few contrast, etc). The main challenge for maintaining, evolving and testing these systems is to deal with the combinatorial explosion of (i) the number of configurations for individual components and of (ii) all different kinds of images the assemblies must handle. Several restrictions do exist when designing these configurations and images that can be captured in constraint models. However, there currently exists no systematic method to generate and qualify test configurations in this context. In this chapter we validate the VANE solution [†1]. .

[†1]This chapter is based in [69] and part of this material has been published in the ISSTA conference

11.1 Evaluation

The acquisition, processing, and analysis of video sequences find applications in our daily life. Surveillance applications, computer-aided medical imaging systems, traffic control, and mobile robotics are some examples. Such software-intensive systems rely on signals captured by video cameras, which are then, processed through a chain of algorithms. Basic signal processing algorithms are assembled in different ways, depending on the goal of image recognition (scene interpretation, follow a specific object, etc.). The algorithms produce numerical or symbolic information helpful for humans or subject to subsequent analysis. For example, rectangles covering the zone of a specific object help tracking people or vehicles in motion.

The MOTIV project aims at evaluating computer vision algorithms such as those used for surveillance or rescue operations. A targeted scenario is usually as follows. First, airborne or land-based cameras capture on-the-fly videos. Then, the processing and analysis of video sequences are performed to detect and track, for example, survivors in a natural disaster. Eventually, and based on the information, the operations manager can achieve strategic or tactical goals in a rapid manner. Two organizations are part of the MOTIV project as well as the DGA (the French governmental organization for defense procurement). The two companies develop and provide numerous algorithms for video analysis. Clearly, there is no one-size-fits-all solution capable of handling the diversity of scenarios and signal qualities. This poses a difficult problem for all the partners of MOTIV: which algorithms are best suited given a particular application? From the consumer side (the DGA), how to choose, select and combine the algorithms? From the provider side (the two companies), how to guarantee that the algorithms meet a large variety of conditions? How to propose innovative solutions able to handle new situations?

The fundamental and common challenge is the *testing* of algorithms. All the partners, being providers or consumers, aim to determine what algorithms are likely to fail or excel in certain conditions. Empirical and statistical methods (based on numerous metrics for assessing non-functional dimensions such as performance or reliability) have been developed and are intensively used for this purpose. Nevertheless, practitioners face severe difficulties to obtain an input test suite (i.e., a set of video sequences) large and diverse enough to test the algorithms. The current practice is indeed to find some existing videos or film video sequences outside. The effort of manually collecting videos is highly consuming in time and resources. First, high costs and complex logistics are required to film video sequences in real locations. Second, the ground truth should be elaborated for every video sequence – it is again

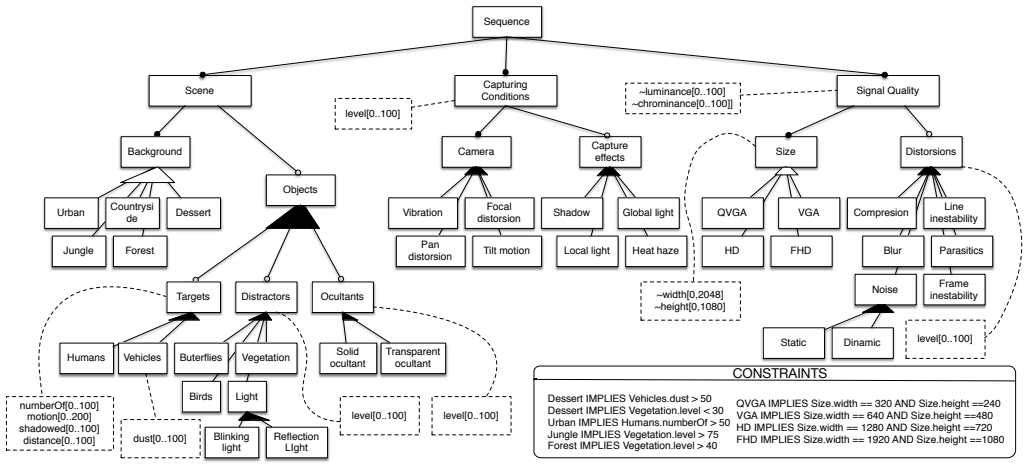


Figure 11.1: *Feature model to represent variability of a video sequence.*

time-consuming and also error-prone. Due to the practical difficulties, the number of collected video sequences is too low and the videos are not different enough to test algorithms. In addition, practitioners have limited control over the scenarios covered by the set of video sequences. As a result, the major challenge for testing the algorithms remains: *How to obtain a suitable and comprehensive input set of video sequences?*

We aim to evaluate our variability-based testing approach along two dimensions:

- **Scalability of VANE.** As described in Section §11, very large sets of testable configurations are possible. An implementation of VANE should be able to cope with models encoding large numbers of features and attributes. Specifically, we aim to evaluate how the approach scales when deriving T-wise configurations using the large-scale and realistic feature model of the MOTIV project. We also aim to determine the number of configurations required when optimizing different values of the feature model.

- **Practical benefits and limits.** of the approach in the context of an industrial project (see Section §11 and the MOTIV project). The introduction of variability modeling and testing techniques aims at improving current practice. We discuss qualitative properties of the approach as part of an action-based research we conduct.

Hypotheses of VANE first experiment			
Null Hypothesis (H_0)	The time required by VANE when deriving t-covering sets is not affordable for most cases.		
Alt. Hypothesis (H_1)	VANE effectively optimize different functions when using attributed feature models describing hight configurable systems.		
Hypotheses of Experiment 2			
Null Hypothesis (H_0)	The multi-optimization operation does not returns results in a reasonable amount of time		
Alt. Hypothesis (H_1)	VANE derives multi-objectives functions covering sets in a reasonable amount of time.		
Design			
Dependent Variable	Hardware options covered by the tests		
Blocking Variables	Maximum cost allowed	**Levels:**	5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70
Constants	CSP solver	**Value:**	ChocoSolver [†2]
	Heuristic for variable selection in the CSP solver	**Value:**	*Default*
	Model used as input	**Value:**	*Android feature model presented in Figure §10.2*

Table 11.1: *Hypotheses and design for the VANE solution.*

Table §11.1 present the different hypothesis we assumed when performing the experiments presented in this section.

11.1.1 Applicability of VANE to a real-scenario.

Experimental data and platform. we used the feature model of the MOTIV project to test the scalability of VANE. The feature model was elaborated during several meetings with the MOTIV partners. Their expertise in the development and quantification of (embedded) video processing systems helped us to know more about the video domain. It should be noted that each configuration of a feature model can be translated to a configuration file (through assignment of values to attributes)(see Figure §7.2). The design of the configuration file helps to design realistic and concrete correspondences at the implementation level.

An excerpt of the resulting feature model is shown in Figure §11.1. This

feature model contains a large set of attributes and constraints that reduces the total number of permutations between input combinations. For example, this model prevents us of generating video-sequences having more than 50% of dustAmount when using an mobile platform featureUrban background. It should be noted that in Figure §11.1, we mark the non-inheritable attributes with the ~ mark – the rest of attributes are held by their associated feature and its children.

Our experiment have been tested using a Java implementation of VANE. This implementation is provided as part of the FAMILIAR tool [4]. Internally, this solution uses the Choco CSP solver. The experiments were performed on a Intel Core i5 running at 1.5, 8 GB of RAM memory, the 1.7 Oracle java virtual machine and Os X v10.9.1 as operating system.

Hypothesis: VANE derives pair-wise sets in an affordable time when optimizing a concrete value. One of the main aspects of the VANE solution is the ability of maximizing or minimizing custom functions over attributes and features. We measured the time and the number of different configurations required to obtain a pair-wise coverage for each quality attribute in the feature model. Our hypothesis is that VANE is able to derive optimal pair-wise configuration sets for video-sequences in an affordable time. In this experiment we are only considering the maximization of one attribute per execution. That is, each time we call the CSP solver, we ask for the variable assignation maximizing a function which is the value of one attribute. We consider that the VANE solution must derive pair-wise combinations in less than 15 minutes to determine if the VANE is valid for the purposes of the MOTIV project.

Results. Figure §11.2 shows the time required by VANE when optimizing each attribute present in the model. In this figure, the vertical axis shows the number of seconds required to optimize the attribute in the horizontal axis. The results show that none of the optimization operations take more than 15 minutes to finish. Moreover, the time vary from the 171.64 seconds (Vehicles.distance) to 680,75 seconds (TiltMotion). The time required to generate a video-sequence manually is largely higher than 30 minutes, thus, we think that spend 5 ~ 15 minutes is worth to reduce the number of video sequences to generate. Nevertheless, Figure §11.3 shows the number of configurations required to use when optimizing each attribute. The number of pairs to cover is 767, thus, by using VANE it was decreased by at least 269 configurations. This amount of configurations is obtained not by minimizing the number of configurations but only when maximizing an attribute. This is, for this experiment, we did not minimize the number of configurations to use and the testing cost reduction is very noticeable.

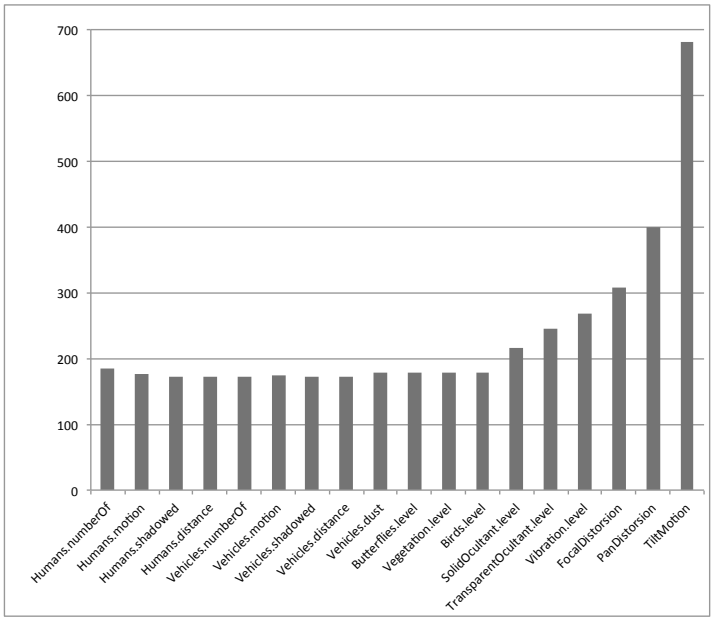


Figure 11.2: *Required time when VANE optimizes one quality attribute.*

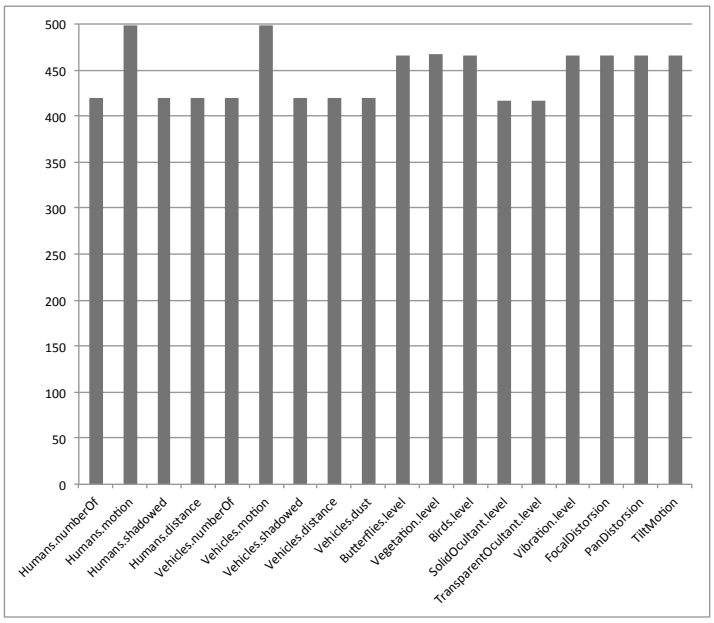


Figure 11.3: *# configurations optimizing one quality attribute.*

11.1.2 Multi-objective capabilities evaluation.

VANE has multi-objective capabilities that enable the optimization of more than one function at the same time. Section §7.3.4 explains how is it implemented by using an evolutionary algorithm that internally uses a CSP for calculating the fitness function. In this experiment we try to minimize the number of configurations required to grant the pair-wise coverage maximizing one attributed in a set of thirty random models. The random models were created by using the Betty tool [156] which implements Thum et al.[174] algorithms. In the random set of models, we fixed the number of features to ten and vary the percentage of cross-tree constraints from 5% to 15% of cross-tree constraints. All models have two attributes per feature and two complex constraints. Also, we specified a timeout of 1 minute for each evaluation of the fitness function. This is, the CSP returns the best solution found in 1 minute. The stop criteria used is to finish the execution after two generations without improvement or when reaching the twenty-fifth generation.

Hypothesis: Multi-objective scalability. Our hypothesis is that VANE effectively reduce the number of configurations needed to execute when using pair-wise reduction at the same time that maximizing an attribute in a reasonable amount of time.

Experiment 2 Results. We found that VANE derives Pareto optimal configurations in no more than 122.34 seconds being the average of the thirty generated models 38.46 seconds. Also, none of the thirty models reached the maximum number of generations. This is, all models required less than twenty-five generations to finish. Moreover, twenty-four out of the thirty models needed only three generations to reach a Pareto solution. The model that required more generations to reach a solution only needed 7 generations.

The experiments presented in this section show: i) the amount of time required by VANE to optimize the set of configurations to execute; and ii) the reduction in terms of the number of configurations. We conclude that the execution time of the tool is worth for the application of pair-wise testing in video-sequences generation. As previously shown, the reduction of costs carried out when generating video-sequences is noticeable.

Scalability and multi-objective. As part of the experiment, we observed that the complexity of *multi-optimization* problems applied for T-wise covering sets carries out high a memory and CPU consumption. For example, the time required to use multi-objective functions in our feature model can take about twelve hours (the previous experience only considers one objective function; more details about the multi-objective results can be found online).

The practical impact is limited though, since experts compute once and for all the configurations; then the variants of video sequences are synthesized offline (see the next section for a discussion about other aspects of the approach). We admit that our current solution prevents an application in contexts requiring fast responses such as it happens in dynamically re-configurable systems.

We have learned the following important lessons:

i. **Scoping the T-wise covering sets.** Different T-wise covering sets maximizes different properties of the test-suite. We learned that the maximization of the different properties can the test-suite for the detection of errors.

ii. **T-wise multi-objective optimization is costly.** We observed that the complexity of multi-optimization problems applied for t-wise covering sets carries out a hight memory and CPU consumption. For example, the time required to use multi-objective functions in our feature model takes about twelve hours. This prevents our current implementation of being applicable in contexts requiring fast responses such as embedded systems.

11.2 Benefits and limits of the approach

We discuss the approach in the context of the MOTIV project. The project is still ongoing, but we already observed some practical benefits and limits; we also identified some open questions. Our report is based on discussions we had with the partners in nine formal and informal meetings (i.e., workshop sessions and visits to companies).

Automation. The feature model used in the MOTIV project encodes more than 240 different Boolean parameters among other quality attributes. Compared to a manual approach, VANE selects around 500 pairs that maximize a custom attribute. This amount of configurations cannot be handled properly by a manual approach. As stated in Section §11, preparing a test case (video sequence) requires to film or find out a realistic video, to prepare the ground truth, etc. The MOTIV partners estimate that the preparation of a test case requires at least 2 hours of work per men. In practice, handling five hundred configurations is clearly unfeasible. Another benefit of automation is to reproduce the synthesis with other parameters for targeting more specific scenarios.

Covering. Without a proper and explicit variability modeling, practitioners ignore what test cases are actually handled and covered. The knowledge of

the coverage is very important since most situations are covered, thus, practitioners are more confident in terms of robustness, performance and reliability of the algorithm. It is especially important for an institution like DGA to have such a strong coverage guarantee. It is as important for the two industrial partners to cover the maximum kinds of situations and determine if the algorithms behave accordingly. VANE grants, by construction, the validity of the T-wise configuration set while enabling multi-objective testing optimization. Another argument in favor of combinational (like T-wise) and optimization techniques is that the number of testable configurations should not be too high. Indeed the synthesis of a video sequence variant is resource consuming and can take 1.5h according to the MOTIV partners[†3]. This is an additional reason that motivates the reduction of the number of tests through combinational covering.

Flexibility. The VANE approach enables practitioners to control the synthesis of T-wise covering sets. Traditionally, testing approaches (mainly manual or exploratory) rely on the expertise of the practitioners. Different alternatives can be employed for this purpose:

- Putting additional constraints and specializing the feature model for specific scenarios. For instance, a specific mobile platform featureBackground (e.g., mobile platform featureUrban) can be set up since the application is known to be deployed in a specific military ground. In turn, the testing machinery will then consider only configurations with mobile platform featureUrban. The benefit is that practitioners can focus on specific testing scenarios, specializing the test suite to realistic cases.
- Optimizing different objective functions over attributes: practitioners can specify the relative importance or cost of a feature, fix some parameters, etc. Again, it aims at customizing the test suite to fit realistic needs.

In all cases, VANE provides the flexibility of optimizing parameters without altering the coverage of feature combinations (because of the T-wise criterion).The VANE implementation and experiments described in this chapter are available in open source form as part of the FAMILIAR tool. Moreover, we provide the experiments inputs and the results https://github.com/ViViD-DiverSE/Experiments.

Realisticness. The major threat when synthesizing video sequences is to produce unrealistic video sequence or videos that present limited interest when testing algorithms. For instance, some natural scenes may not be

[†3]As part of our experiment we did not measure the time needed for synthesizing video sequences – we stop at the generation of configurations.

reliably recreated or the global luminosity eases too much the tasks of an algorithm. Until now, we have not encountered such situations: the variants of video sequences reviewed by the experts so far have been assessed as realistic and ready to test algorithms. However, the experts have not reviewed the *entire* test suite. It is still unclear how a reviewing process could look like and how we can ease that task – pointing out to experts what are the features activated in the video sequence seems an interesting option. In case of detecting an unrealistic variant, practitioners can add some constraints and exploit the *flexibility* of the approach (see above).

Exploration and incremental synthesis. Another promising direction is to allow practitioners to explicitly report on unrealistic variants. Also, the testing techniques should be revised/adapted to take this information into account, i.e., so that the test suite no longer contains an unrealistic video sequence. More generally it would enable an incremental synthesis process where practitioners modify objective functions, attributes, and constraints of the feature model to improve the suitability/realisticness (if needs be) of the video sequences.

Effort and reuse. The effort needed to realize the approach mainly consists in i) elaborating a variability model, ii) selecting a ”base” video sequence, and iii) developing video processing functions to modify elements (e.g., luminosity) or inject new elements (e.g., a tank). Our observation is that the major effort resides in the third step. It is unclear, at this step of the research, to determine if the effort pays off and can be reused for other ”base” video sequences.

11.3 Threats to validity

Even though the experiments presented in this chapter provide evidence that the solution proposed is valid, there are some conditions that may affect their validity.

External validity. The inputs used for the experiments presented in this chapter represent a realistic feature model. We consider that the feature model is realistic since numerous experts were involved in the design. Moreover, the model has proven to have an implementation counterpart (the configuration files) useful for synthesizing variants of video sequences. However, it is possible that the feature model do not reflect properly the same structure as other realistic models. The major threats to the external validity are: *population validity*, since the model used in the experiment represent only one concrete

instance of the problem (there might be other models that do not mirror the model presented in Figure §11.1); *ecological validity*: VANE analysis were executed 10-times and we report on averages to minimize the impact of third-party threads in the time being measured.

Internal validity. The CPU capabilities required when analyzing a feature model depend on the number of features, percentage, and nature of cross-tree constraints. However, multi-optimization and pair-wise derivation add other new variables affecting the performance, such as the number of the pairs to cover. We experimented with multiple variations of quality attributes to limit the internal validity. We plan to vary objective functions as well, based on knowledge and requirements of video experts.

Construct validity. The results looks promising in terms of time required to solve problems related to our feature model. However, we might need to perform a higher scale experimentation when referring to multi-objectives configuration problems.

11.4 Summary

The complexity existing in the video-surveillance algorithms introduces new challenges to face when testing them. Moreover, there are multiple and conflicting functions to optimize at the same time.

In this chapter we have evaluated our tool VANE in this context. First, we encoded the variability existing in a video-sequence generator, later we derived pair-wise configurations that optimize certain parameters and later, we stresed our tool when conflicting functions exists. Moreover, this approach is currently being used in the context of MOTIV project, more than 5000 videos have been generated to build up a test-suite for video-surveillance algorithms.

Chapter 12

Scalability of feature model drifts configuration.

Do What thou wilt

Die unendliche Geschichte, German people

Configuring an SPL over multiple steps is a highly combinatorial problem. An automated multi-step SPL configuration technique should be able to scale to hundreds of features and multiple steps. This section presents empirical results from experiments we performed to determine the scalability of MUSCLES. We tested a number of hypotheses related to the scalability of MUSCLES using various SPL configuration parameters, such as the total number of configuration steps[†1].

[†1] Part of the material used in this chapter was accepted in the JSS journal[179]

12.1 Experimental platform

Our first experiment was performed with an implementation of MUSCLES provided by the open-source Ascent Design Studio (available from code.google.com/p/ascent-design-studio). The Ascent Design Studio's implementation of MUSCLES is built using the Java Choco open-source CSP solver (available from choco.sourceforge.net). The experiments were performed on a computer with an Intel Core DUO 2.4GHZ CPU, 2 gigabytes of memory, Windows XP, and a version 1.6 Java Virtual Machine (JVM). The JVM was run in server mode using a heap size of 40 megabytes (-Xms40m) and a maximum memory size of 256 megabytes (-Xmx256m).

The second experiment was performed with an implementation of the MUSCLES provided by the open-source FAMA tool suite. FAMA is also built using the Java Choco open-source CSP solver. The experiments were performed on a rack-mounted DELL PowerEdge server with 12 cores, 2GB of RAM, and running Ubuntu. The JVM was run in server mode using a heap size of 40 megabytes (-Xms40m) and a maximum memory size of 256 megabytes (-Xmx256m).

To test the scalability of MUSCLES we needed thousands of feature models to test with, which posed a problem since there are not many large-scale feature models available to researchers. A CSP solver's performance can vary widely, from extremely fast to exponential time, depending on the constraints of a particular problem characteristic. In practice, CSP solvers tend to perform very well. To be thorough, we wanted to test the technique on a large number of models to get an accurate picture of the solving time. To solve this problem, we used a random feature model generator developed in prior work [180]. The feature model generator and code for these experiments is also available in open-source form along with the Ascent Design Studio. The feature model generator takes as input the desired total number of features, maximum branching factor, total number of cross-tree constraints, and maximum depth for the feature model tree.This feature model generator is based on the techniques developed by Thum et al. [174]. The generator produces a random feature model that meets the requirements. We used a maximum branching factor of 5 children per feature and a maximum of 1/3 of the features were in an XOR group.[†2]

We also needed the ability to produce valid starting and ending configurations that the solver could derive a configuration path between. To produce these configurations, we used the CSP technique developed by Benavides et

[†2]XOR feature groups are features that require the set of their selected children to satisfy a cardinality constraint (the constraint is 1..1 for XOR).

al. [16] to derive valid configurations of the feature model. If the CSP technique could not derive at least two different configurations from the feature model, it was considered void and thrown out.

Our experiments uncovered trends similar to what observed in prior work [180]. In particular, the branching factor, depth, and cross-tree constraints had little effect on configuration time. The key indicator of the solving complexity was the number of XOR-feature groups in a model. The other key indicators of solving complexity where whether or not optimization was used and the total number of time steps involved in the configuration.

12.2 Multi-step configuration scalability

Hypothesis. We hypothesized that MUSCLES could scale up to hundreds of features and 10 or more time steps, having this hypothesis we designed this experiment to prove it, believing that a CSP solver would be fast enough to derive a configuration path in a few seconds. On the other hand, the null hypothesis of this experiment is that the solver can't find a suitable solution for the problem in a reasonable amount of time.

Experiment design. We measured the solving time of MUSCLES by generating random multi-step configuration problems and solving for configuration paths that involved larger and larger numbers of steps. The problems were created by generating semi-random feature models with 500 features as well as starting and ending configurations for each model. The models were obtained using the tool Betty, which provides an implementation of the Thum et al.[174] proposal. The point configuration constraints were derived by using a constraint solver to derive a valid configuration for each step. Once a point configuration was chosen for a step, an edge constraint was added limiting the sum of the feature selection and deselection costs to be exactly the sum needed to reach the randomly chosen point configuration. Table §12.1 shows the details of the experiment executed. MUSCLES was used to derive a configuration path between the starting and ending configurations.

Our experiments were performed with *large-scale configuration paths*, which were produced by forcing the solver to find a configuration path that involved switching between two children of the root feature that were involved in an XOR group. For a feature model with 500 features configured over 3 steps, the worst case solving time we observed was ~3 seconds. The worst case solving time for feature models configured over 10 steps was 16 seconds. These initial results indicate that the technique should be sufficiently fast for

Hypotheses			
Null Hypothesis (H_0)	MUSCLES can rapidly identify the configuration paths from the starting configuration to the final configuration with hundreds features and 10 or more time steps.		
Alt. Hypothesis (H_1)	The complexity of the problem will prevent MUSCLES from finding solutions for problems with hundreds of features and 10 steps.		
Design			
Dependent Variable	Time required by MUSCLES to derive a configuration path		
Independent Variable I	Technique used for FM generation	**Levels:**	Random models produced by Thum et al.'s technique[174]
	Technique for selecting point and edge constraints	**Levels:**	Random (Java function applied to the list of model features traversed inorder.)
Blocking Variables	Number of Steps	**Levels:**	3,4,5,6,7,8,9,10
Constants	CSP solver	**Value:**	ChocoReasoner
	Heuristic for variable selection in the CSP solver	**Value:**	*Default*
	Percentage of Cross-Tree Constraints ($\frac{constraints}{features}$)	**Levels:**	%10

Table 12.1: *MUSCLES first experiment data.*

feature models with hundreds of features.

Figure §12.1 shows an example of a large-scale configuration path problem where the solver must derive a configuration path that switches from including feature `A` to feature `B`.

With this type of configuration problem, the solver was forced to change every feature selection in the starting configuration to reach the end state, *i.e.*, these experiments maximized the difference between the starting and ending configurations.

We generated and solved temporal configuration path problems for feature models with 500 features. We successively increased the number of time steps involved in the configuration path to produce larger and larger configuration paths. The maximum number of changes per configuration checkpoint were bounded to 1/4 of the total number of features. We solved 100 randomly

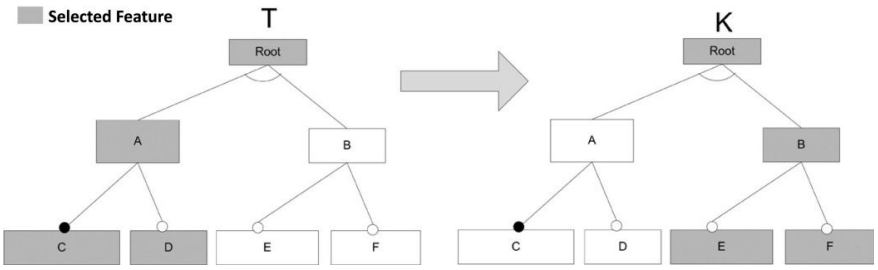


Figure 12.1: *Changing between two XOR subtrees.*

generated configuration path problems per feature model size.

Results and analysis. The results from the experiment are shown in Figure §12.2.

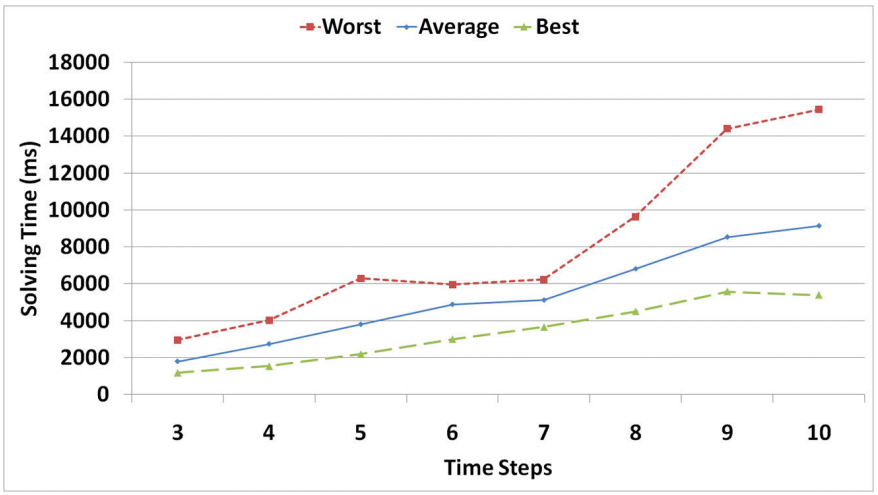


Figure 12.2: *Automated configuration time for varying numbers of time steps.*

This figure shows the solving time in milliseconds for the configuration path derivation versus the total number of time steps in the configuration problem. As shown in Figure §12.2, the solving time scales roughly linearly with the number of time steps.

The apparent linear scaling of the technique with respect to the number of time steps is a promising result. Although more work is needed to show that

this linear scaling continues for different configuration path properties, these results indicate that the technique may scale well as the number of time steps grows. Our future work will further investigate the scalability of the technique and improve MUSCLES's CSP formulation. We also note that standard CSP solving algorithms, such as branch and bound appear to work well for these problems even though they have exponential worst case time complexity. However, it may be possible to develop new solving algorithms that provide better performance for special classes of multi-step configuration problems. Based on the results, we ended agreeing that the null hypothesis we proposed have been proved and MUSCLES can rapidly identify the configuration path with hundreds of features and 10 or more steps.

12.3 Feature model drift scalability

Hypothesis. In this experiment we proposed as a null hypothesis, that MUSCLES could solve for configuration paths that included feature model drift in several seconds. Given that null hypothesis, the alternative hypothesis is that MUSCLES can not solve problems with drifts in a few seconds.

Experiment design. As in the first experiment, we measured the solving time of MUSCLES by generating random multi-step configuration problems and solving for configuration paths that involved larger and larger numbers of steps. In this second experiment, we introduced changes to the feature model at each step. At each step, one feature was added or removed. The feature model was then checked to ensure that it included one or more valid products using CSP analysis. If the new feature model did not contain any valid products, the feature change was reversed and another random change attempted. The feature models were semi-randomly generated with 20-2000 features as well as starting and ending configurations for each model. MUSCLES was used to derive a configuration path between the two configurations over multiple steps. The properties of the feature models described in Experiment 1 were also used for this experiment.

Table §12.2 shows the details of Experiment 2.

Results and analysis. The results from the experiment are shown in Figure §12.3.

This figure shows the solving time in milliseconds for the configuration path derivation versus the total number of features. Overall, the approach scaled well for large feature models. At 1,000 features, a solution could be found in 4 seconds or less. We believe that for the majority of industry prob-

Hypotheses			
Null Hypothesis (H_0)	MUSCLES can find solutions for problems that incorporate feature model drift and have a high number of features in several seconds.		
Alt. Hypothesis (H_1)	MUSCLES can not solve for solutions within a few seconds if feature model drift occurs.		
Design			
Dependent Variable	Time required by MUSCLES to configure a model with varying drifts.		
Independent Variable	Technique used for FM generation	**Levels:**	Random models generated using Thum et al.'s technique[174]
	Technique for selecting point and edge constraints	**Levels:**	Random (Java function applied to the list of model features traversed inorder.)
Blocking Variables	Number of Features	**Levels:**	20, 100, 200, 500, 1000, 2000
Constants	CSP solver	**Value:**	ChocoReasoner
	Heuristic for variable selection in the CSP solver	**Value:**	*Default*
	Percentage of Cross-Tree Constraints ($\frac{constraints}{features}$)	**Levels:**	%10
	Number of steps	**Levels:**	3
	Selection of initial and final configurations	**Levels:**	Random (from the set of valid configurations)

Table 12.2: *MUSCLES second experiment data .*

lems, being able to deal with feature models with 1,000 features will be sufficient. Therefore, given those results, the null hypothesis proved to be correct.

12.4 Threats to validity

Even though the experiments presented in this chapter provide evidence that the solution proposed is valid, there are some conditions that may affect the validity of those experiments. In this section we show the different validity threads that could affect the experiments.

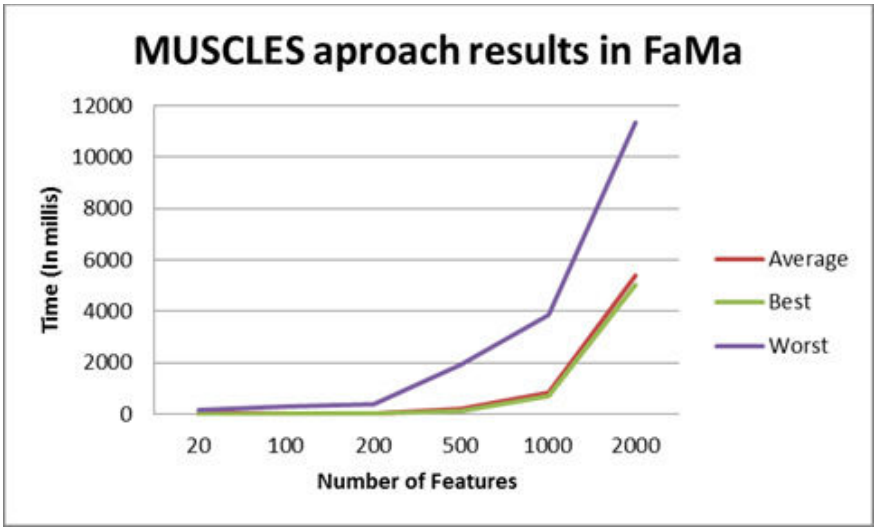


Figure 12.3: *Automated configuration time for feature model drift problems.*

External Validity The inputs for the experiments presented in this chapter have been inspired by industry problems. However, it is possible that the feature models that we have experimented with do not properly reflect real-world models. The major threats to the external validity of are experiments are: 1) **Population validity**, the models used are randomly created and may not mirror realistic feature models seen in industry. The complexity of the constraints used and the size of the problem may vary with real projects. To try to minimize this effect we have relied on the Thum et al.'s feature model generation approach [174] and its implementation provided by the BeTTy tool [156]. 2) **Ecological validity**, MUSCLES analyses were run individually to minimize the impact of third-party threads in the time being measured. However, there might be other threads such as operating system threads, that could impact execution time. To minimize this effect, we carried out not one execution of a model, but a hundred executions and used the average as the result.

Internal Validity The time required to analyze a feature model depends on the number of features and percentage of cross-tree constraint and deriving SPL configurations has been proven to be an NP problem in previous research [19, 91]. Multi-step configuration problems add other inputs that might affect the performance, such as the number of the steps required to reach the final configuration. The conducted experiments were designed to not exceed a maximum budget between successive

steps. However, if we add attributes to the experiments, more complex functions than the sum of costs by selecting and deselecting features can be employed. For example, a more complex function over the numeric delta between attributes of successive steps could be used, such as combining carbon emissions with monetary costs. To improve the internal validity of the experiments we experimented with multiple variations on the number of features and a variety of step counts.

Construct Validity The first results looks promising in terms of time required to solve problems with 1,000 features. We assume that most real-world problems will be of similar scale. However, because the tests were not exhaustive, more analysis of which solver heuristics provide the best results are needed.

12.5 Summary

Variability intensive systems change rapidly, therefore, mechanishms granting the safe transition between different evolution phases are required. In this chapter, we have shown the scalability of MUSCLES. Concretely, we have evaluated the multi-step configuration process and the feature model drift technique.

The Ascent Design Studio (http://ascent-design-studio.googlecode.com) and FAMA (http://famats.googlecode.com/svn/branches/multistep) provide open-source implementations of MUSCLES.

Chapter 13

Validating Invar

Dar más vueltas que el tío los caballitos

Dicho popular, Andalusian people

In this section we demonstrate that Invar *is flexible enough to allow the integration of different variability modeling approaches. We also evaluate the conceptual integrity of our approach by applying it to three different realistic configuration scenarios to address research question Finally, we present a performance assessment to know what is the impact of the* Invar *enactment strategies on performance*[†1].

[†1]Part of this material have been published in the SPLC conference[56] and the VAMOS workshop [55]

13.1 Integrating three different variability modeling approaches

We have so far implemented the Invar service configuration interface for the feature-oriented FaMa tool suite [20], the OVM-oriented FaMa tool [145, 146] and the decision-based DOPLER [54] tool suite. We chose these approaches as we have gained several years of experience of applying them in academic and industrial settings including large-scale product lines [19, 21, 32, 54, 139, 142]. Furthermore, these three approaches represent three distinct classes of modeling techniques in SPLE.

13.1.1 Plugging feature models to Invar

The FaMa approach [20] supports different kind of feature model dialects and allows using different solvers in the back-end to perform analysis operations on feature models. Currently it implements analysis using constraint programming, SAT and BDD solvers. Other solvers can easily be plugged–in. FaMa also provides capabilities to automatically test new implementations [158].

The *implementation* of Invar configuration services for FaMa faced several issues as FaMa was not designed to be used for questionnaire-based product configuration. Also, FaMa was primarily implemented as a framework to perform automated analysis of feature models, i.e., it eases the automated extraction of information from feature models and not configuration based on feature models. Nevertheless, the adaptation to Invar Web Service interfaces was almost straightforward – a good example demonstrating the flexibility of Invar. The key mappings between the Invar configuration steps and FaMa can be summarized as follows:

Question Types: For the sake of simplicity, here we only considered feature models with four kinds of relationships, which were mapped to Invar question types as follows: A mandatory relationship in a feature model is a relationship between a parent and a child where the child has to be selected whenever the parent is selected. Hence, in this case, no question is asked to the user, as their is no choice anyway. An optional relationship between a parent and a child means that the child can be selected or deselected whenever the parent feature is selected. We mapped an optional relationship to an Alternative question type in Invar with only one option, this is, a single check box. An or-relationship between a parent and a set of children determines that at least one child has to be selected whenever the parent is selected. Any combination

of children is also allowed. We mapped the or-relationship to a MoreThanOne question type in Invar with multiple check boxes. An alternative relationship between a parent and a set of children determines that one and only one child has to be selected whenever the parent is selected. An alternative relationship maps to the Alternative question type in Invar.

Order of Questions: Feature models are not designed for workflow-oriented configuration. Hence, there is no predefined order of presenting questions to the end user. In our implementation we decided to define as default to traverse the tree in a pre-order-like style. The FaMa FM implementation provides (i) three traversal-like orders (in-order, pre-order and post-order); (ii) two orders based on the variability in the models, i.e., the first one starting with the questions with most options and ending with the questions with few options, and the second one in the opposite order, from few options to the most; and (iii) an out of the box way to traverse the questions by name, using an alphabetical order (cf. Section §9.2.4.1). The user can also define an add–hoc order by manually sorting the list of questions on features.

Feedback: In FaMa, at any time a configuration can give feedback to the Invar configuration service. The feedback supported includes: (i) informing whether a given feature is selected or deselected, (ii) determining whether the current configuration is valid, i.e., it is possible to extend the configuration to a valid product, (iii) calculating the total number of potential configurations of the model, (iv) informing about the number of questions that have not been decided yet, (v) calculating the number of potential configurations available according to the current selection/deselection of features, and (vi) determining whether the current configuration is valid as a final product. A configuration is valid if all the features involved are either selected or deselected and if there are no conflicts between the existing constraints in the models and the actual state of the features, meaning that for each feature it is clear whether via the feature state whether it is in the configuration or not.

13.1.2 Plugging OVM models to Invar

FaMa-OVM [145] is an instance of the FaMa framework [21] but using the OVM notation. The same analysis capabilities and extensions can be supported as for feature models (cf. Section §13.1.1). The OVM modeling approach was not designed to be used in a questionnaire-based configuration process so we had to face some issues when *implementing* the FaMa-OVM configuration service for Invar. However we solved the problems as follows:

Question Types: Internally, FaMa-OVM supports the usage of OVM as de-

Feature Modeling	Invar
Feature	*Option*
Mandatory subfeature	(ignored, feature will be selected in any case)
Optional subfeature	*Alternative* with only one option (one checkbox)
Or group	*MoreThanOne* (multiple checkboxes)
Alternative group	*Alternative* with multiple options

Table 13.1: *Mapping of feature model elements to* Invar *primitives.*

scribed by Pohl *et al.* [135]. In Invar we take into account *mandatory*, *optional*, *alternative* and *or* relationships as they offer a straight way to map OVM to Invar. A mapping of a mandatory relationship between a mandatory VP and a V is not necessary, since this does not define any variability to be configured. If the mapping of the mandatory relationship is between an optional VP and a V, then we map that relationship as an Alternative question type in Invar. Every optional relationship is mapped as Alternative question type in Invar with only one option. Every or-relationship is mapped as a MoreThanOne question type in Invar. An alternative relationship maps to the Alternative question type in Invar.

Order of Questions: In our implementation we use a random order as a default option to traverse every VP in the model. However, as Invar offers the ordering mechanisms presented in Section §9.2.4.1, we also allow the FaMa-OVM Invar plugin to offer an alphabetical order and two orders based on the amount of variants that every variation point has, one from more-to-less and another one from less-to-more variants.

Feedback: FaMa-OVM allows to analyze OVM making it possible to give feedback to the user. More specifically, FaMa-OVM returns to Invar the same feedback as FaMa FM, i.e., (i) when a VP or a V is selected/deselected; (ii) when the configuration is valid; (iii) if the actual configuration is valid as a final product; (iv) how many valid configurations exist in the model; and (v) how many questions are remaining to finish the configuration.

13.1.3 Plugging decision models to Invar

The DOPLER approach [54] allows defining decision models together with the reusable assets of a product line (e.g., reusable software components) and

Orthogonal Variability Modeling	Invar
Variant	Option
Variation point	Option
Mandatory relationship	(ignored, variant will be selected in any case)
Optional relationship	*Alternative* with only one option (checkbox)
Or relationship	*MoreThanOne*
Alternative choice	*Alternative*

Table 13.2: *Mapping of OVM model elements to* Invar *primitives.*

mappings between the assets and the decisions. A domain-specific metamodel defines the possible types of assets, their attributes, and dependencies. In addition to hierarchical dependencies among decisions, other dependencies (comparable to cross–tree constraints) are modeled using rules of the form *if condition then action* [54]. The DOPLER tool suite [54] uses a Java-based Rule Language (JRL) and execution engine as a back-end for evaluating the rules defined in models. For a description of diverse application examples refer to [54, 141].

The *implementation* of the DOPLER decision modeling approach to provide Invar configuration services was straightforward, as the DOPLER approach itself was designed to be used for questionnaire-based product configuration [140]. The mapping from Invar to DOPLER in many cases only required calling the respective method in the DOPLER API. Some of the key mappings between the Invar configuration steps and DOPLER can be summarized as follows:

Question Types: We had to map the Invar question types to DOPLER decision types. DOPLER decision types are Boolean, String, Number and Enumeration. Enumeration decisions can be defined with a cardinality defining the subset of the set of possible answers to the decision that might be selected (e.g., 1:1, 1:n, 2:6). For the sake of simplicity, we have only implemented the mapping for Boolean and Enumeration decision types. This is sufficient as String and Number decisions can also be presented as an Enumeration decision with one option (being a string or a number). More specifically, we mapped the DOPLER Boolean decisions to the Alternative question type with the options yes or no, the DOPLER enumeration decisions with cardinality 1:1 or 0:1 were also mapped to the question type Alternative (with the enumeration literals as options), and the DOPLER enumeration decisions with all other possible cardinalities were mapped to the Invar question type Optional.

Order of Questions: In DOPLER, the order of making decisions is defined by the decisions' dependencies. Top level decisions (which are not dependent on other decisions) are presented and answered first. Decisions which directly depend on top level decisions can be answered next, and so forth. In addition to these hierarchical dependencies, DOPLER allows defining logical dependencies that cross–cut the hierarchical tree structure. For example, answering a particular decision might require changing the value of another decision located somewhere else in the hierarchy. In the Invar configuration service interface, the methods getFirstQuestion(), getNextQuestion() and getPreviousQuestion() implement the navigation strategy. When initializing a DOPLER decision model with Invar, first a sorted list is built based on the decision hierarchy. This list is frequently updated whenever new decisions are added or the order of decisions is changed due to some rule defined in the DOPLER decision model. The order of decisions on one level (e.g., top level) is randomly defined. Logical dependencies are currently not considered by the first/next/previous methods because they would require "jumping" within the model, which might confuse the end user. Whenever making a decision which has an effect on another decision, this effect has to be presented separately, e.g., by informing the user that the other decision was changed and asking her whether she wants to navigate to that decision. Invar allows to define custom non–predefined orders. For DOPLER we do not delegate any order options to the Invar ordering component, as DOPLER models have been created keeping in mind a configuration process and has its own mechanisms [54] to define the order as described above.

Feedback: In DOPLER, making decisions leads to the inclusion and/or parametrization of assets related to the decisions. For example, selecting the document management solution by answering a decision question leads to the inclusion of the respective software component(s) implementing document management. Answering a lower-level decision (e.g., on which type of scanner is required) parameterizes the document management software component. Feedback to the user of the Invar service implementation for DOPLER is thus given by presenting her with the assets that are required for the product currently being configured.

13.2 Configuring the Android privacy settings

To check if Invar applies to different scenarios, we decided to model another domain by using Invar. Concretely we focused in the Android permission systems which has been already proven to be difficult to configure [34]. Moreover, the privacy of mobile phone users have been a trend discussion

Decision Modeling (DOPLER)	Invar
Boolean decisions	*Alternative* with choices "yes" and "no"
Enumerations with cardinality 0:1 or 1:1	*Alternative*
Enumerations with other cardinalities	*Option*

Table 13.3: *Mapping of desicion model elements to* lnvar *primitives.*

over different security foras [63, 64]. Using lnvar we can model the different variability existing in the Android ecosystem so users can configure only applications that meet a certain security criteria. Being able to configure products from diverse models allows us to determine is an application meet the desired security criteria for a concrete user.

In the chapter §10 we show how Android hardware variability can be encoded in a feature model. Now, we model the different traceability relationships in between the Android application structure, the emulator options described in the chapter §10 and the permissions systems in Android. note that in this context, the different models are developed and configured by different entities. For example, the application are configured by the app developers while the security requirements by the app users and the mobile platform configurations to execute the app by different mobile phone manufacturers (or the android emulator designers in our case).

This validation looks forward refuting the null hypothesis *"*lnvar *is not suitable for a diversity of configuration scenarios"*. In the case of not being able to refute it we would need to accept the alternative hypothesis that asserts that lnvar can be used in different real-world scenarios.

13.2.1 The Android permissions system

Android is based in Linux, thus providing a privilege-separated access system[†2]. This is, each application runs with different identity (user/group ID). Also, different parts are also separated, thus, separating applications from each others and from the operating system itself.

A finer-grained security access is also provided through a "permission" mechanism that enforces restrictions on the specific operations that a particular process can perform. Table §13.4 shows the set of Android permissions

[†2]http://developer.android.com/guide/topics/manifest/uses-feature-element.html

Hardware feature	Permission name	Implies This Feature Requirement
Bluetooth	BLUETOOTH	android.hardware.bluetooth
	BLUETOOTH_ADMIN	android.hardware.bluetooth
Camera	CAMERA	android.hardware.camera
		android.hardware.camera.autofocus
Location	ACCESS_MOCK_LOCATION	android.hardware.location
	ACCESS_LOCATION_EXTRA_COMMANDS	android.hardware.location
	INSTALL_LOCATION_PROVIDER	android.hardware.location
	ACCESS_COARSE_LOCATION	android.hardware.location.network
		android.hardware.location
	ACCESS_FINE_LOCATION	android.hardware.location.gps
		android.hardware.location
Microphone	RECORD_AUDIO	android.hardware.microphone
Telephony	CALL_PHONE	android.hardware.telephony
	CALL_PRIVILEGED	android.hardware.telephony
	MODIFY_PHONE_STATE	android.hardware.telephony
	PROCESS_OUTGOING_CALLS	android.hardware.telephony
	READ_SMS	android.hardware.telephony
	RECEIVE_SMS	android.hardware.telephony
	RECEIVE_MMS	android.hardware.telephony
	RECEIVE_WAP_PUSH	android.hardware.telephony
	SEND_SMS	android.hardware.telephony
	WRITE_APN_SETTINGS	android.hardware.telephony
	WRITE_SMS	android.hardware.telephony
Wi-Fi	ACCESS_WIFI_STATE	android.hardware.wifi
	CHANGE_WIFI_STATE	android.hardware.wifi
	CHANGE_WIFI_MULTICAST_STATE	android.hardware.wifi

Table 13.4: *Hardware features used by Android permissions.*

related to the hardware features it requires. We will be using these relations in between features and permissions to enable the configuration of valid products according to a certain security aspects.

13.2.2 Modeling the Android apps

Android has been built to dynamically adapt the applications developed for the operating system to different mobile platform configurations with different screen resolutions, and connectivity capabilities. However, as each application is developed by a different developer (or a set of), the structure itself can differ in between different apps. We here present an OVM model that encode the different variation points existing in an Android app.

Manifest declaration. Each Android application contains a file called MANIFEST.xml where all the different permissions required by the app are specified. Also, in this file, the actions, activities, services and others are specified [†3].

From the set of variation points described in the manifest.xml we considered the following ones:

- **Compatible-screens.** This section describes the different screens supported by an Android application. For defining a compatible screen we need to provide the resolution and dpi of all bitmaps artifacts used in our app.

- **Uses-sdk.** This section present the minimum Android SDK required for the application to work in. This refers to the minimum Android version to execute the application in.

- **Uses-feature.** Android apps may use different and diverse features present in mobile platform configurations. Those features should be included as references in the manifest.xml. An exhaustive list of all existing features in the ecosystem can be found at http://developer.android.com/guide/topics/manifest/uses-feature-element.html.

- **Uses-permisions.** Android apps may use different permissions. Those permissions needs to be declared explicitly in the manifest. Later on this chapter we will define different IMD between uses permissions and hardware features so we can reason and configure products that satisfy our security requirements.

To model this variability existing in the manifest.xml file we used OVM as a notation. Figure §13.1 shows the different variation points existing in the android manifest.xml. Please note that not all the information existing in the manifest.xml is presented in the Figure but only the ones related to the permissions system.

It is important to remark that the different intra-model relationships existing in the Figure §13.1 have been obtained from the permissions descriptions. This is, if in the manifest we declare that we use the feature *android.hardware.bluetooth* in our application it is mandatory to ask for the BLUETOOTH permission. Also, note that these relationships are shown in the Figure §13.1 with dashed blue lines.

[†3] http://developer.android.com/guide/topics/manifest/manifest-intro.html

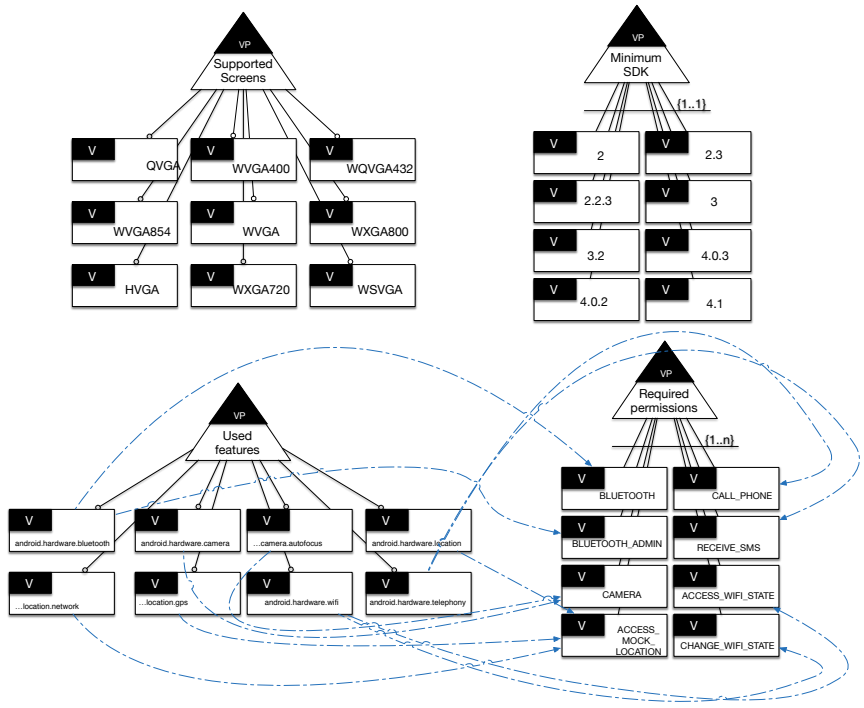


Figure 13.1: *OVM encoding part of the Android application variability.*

13.2.3 Modeling security related user preferences using dopler

Moreover, to being able to configure the Android app, we will define a new Doppler model that will be use to pose the different security questions to the user. For example, to determine if the configured product may expose data through the wifi interface.

Figure §13.2 shows the dopler model that inquires about the security concern when configuring the valid apps and mobile platform configurations existing in the Android ecosystem.

13.2.4 Android permissions as inter-model relationships

Now, what is still missing is the definition of the required inter-model relationships. Defining the appropriated inter-model relationships enable Invar to configure applications meeting mobile platform configurations and privacy

i. Would you need to use bluetooth in your company?

   (a) Would you allow users to add/remove bluetooth devices?

ii. Would you need to allow GSM networking capabilities?

iii. Would you need to allow third party apps to know your location?

iv. Would WIFI be an option for you employees?

v. Will you app users need access to the camera?

Figure 13.2: *Dopler questions used to define the security criteria in Android.*

settings. To do it so, we linked the app supported screens and the mobile platform configurations available screens. This is marked with a circled in the Figure §13.3 for the sake of simplicity. Also, we linked the application used API with the emulator supported Android versions. Finally, we linked the permissions with the different security concerns existing in the Dopler model.

Having modeled the whole chain of different artifacts describing the Android permission system forces us to accept the alternative hypothesis which states that lnvar is able to help in a diversity of configuration scenarios.

13.3 Evaluating the performance of different enactment strategies

Additionally to show how different multi product line settings can be supported with lnvar, we also consider the performance validity of our approach. In this section we thus show the time required to configure a set of different random models with lnvar in order to determine which order provides better results. Executing this experiment, we determined the time required for each variable ordering, showing the average time for automatically configuring a model depending on the number of features and constraints. Further, in this section we demonstrate the "stress" our current implementation is able to withstand, by determining the maximum number of features and cross-tree constraints that it can handle without exceeding an affordable amount of time.

In lnvar, the configuration process of a multi product line is carried out by

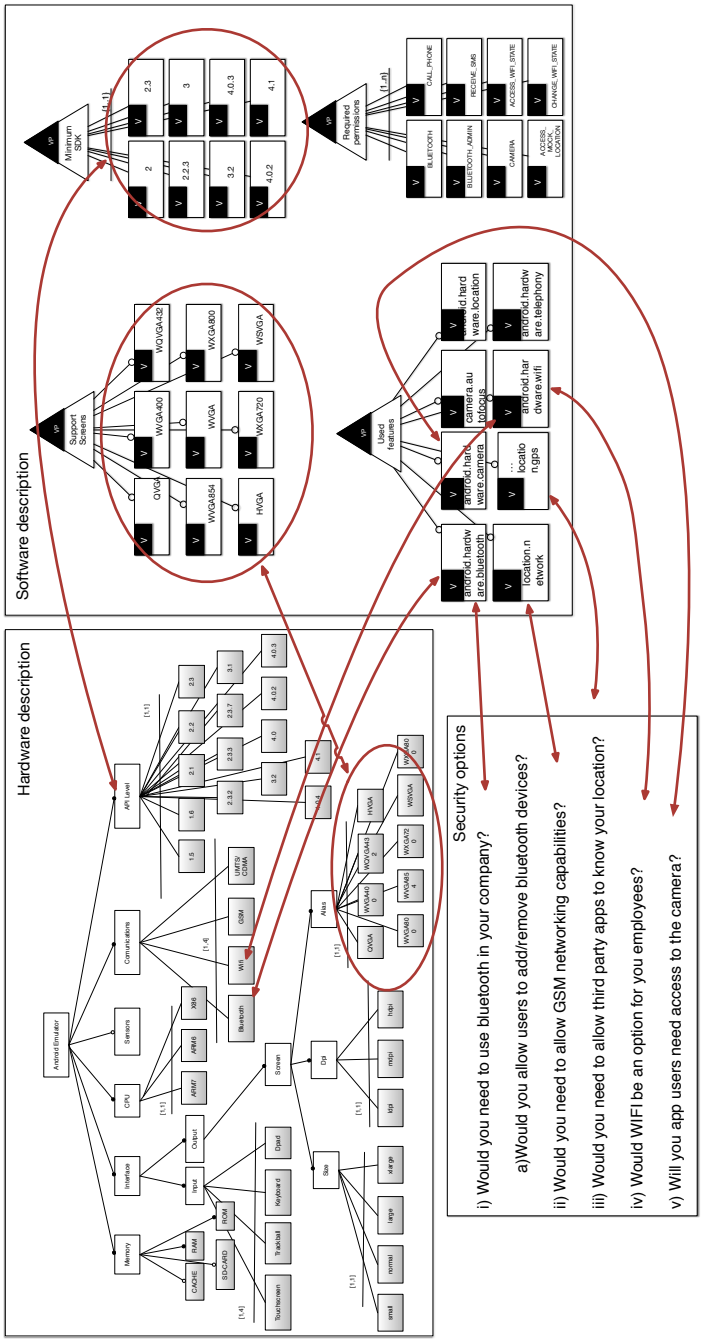


Figure 13.3: *OVM encoding part of the Android application variability.*

selecting or deselecting options from a set of questions. Invar allows to define the order to present those questions to the user in different ways. Mendonca

et al. [111] demonstrated that when analyzing a feature model, the way the solver assigns values to the variables affects to the time required to analyze the model. Our hypothesis is that *the order used to present the options in the configuration process to the user also affects the time needed by the back-end tool to reason over the model.*

To assess whether our hypothesis was correct we designed a set of tests. To execute those tests we selected the FaMa feature model Invar plugin. We selected it because it can be used with SAT or CSP solvers and offers the most orders to be used in an Invar configuration process.

First, we started the testing with the FaMa Invar plugin using a CSP-based solver. We noticed that our CSP implementation for the feature models was not responsive as it should and rarely can reason over models with more than 200 features. Usually, when analyzing a feature model, SAT solvers are more proficient to reason over bigger models than CSP solvers [155]. Therefore, we created a new version of the FaMa Invar plugin, this time using a SAT solver. We conjecture that our results will be similar if we used other solvers than SAT but we wanted to test the approach with larger feature models in order to increase the number of different scenarios where Invar can provide configuration support based on feature models.

To verify our hypothesis we created a set of models to be configured by the FaMa Invar plugin. Those models vary in numbers of features and percentages of cross–tree constraints. Its number of features goes from twenty features to five hundred and from zero percent of cross–tree constraints to twenty percent.

To automate the testing procedure we created an automation tool that acted as the Invar core and the *user* in the configuration process. First an Invar order was selected to perform the testing. Later, this tool created random models using BeTTy [156] and uploaded them to Invar. To generate random models, Betty provides a well-tested implementation for the algorithm presented by Thum *et al.* [174]. After uploading the models, the tool acted as the Invar core and iterated over the questions while randomly selecting or deselecting options. This process was executed in one hundred iterations, yielding the average of the time required. In Table §13.5 we describe the variables used to run our experiments.

Results are presented in Figure §13.4. This figure shows time in milliseconds (vertical axis) needed to configure a model with a given number of features (horizontal axis) and ratio of constraints. Please note that the time is shown on a logarithmic scale.

Hypotheses			
Null Hypothesis (H_0)	The order used in the lnvar configuration process affects the performance of the back-end tool in an lnvar configuration process		
Alt. Hypothesis (H_1)	There is not a significant difference in the average time required by the back-end tools in an lnvar configuration process		
Design			
Dependent Variable	Time required by the back-end tool to configure a model		
Independent Variable	Technique used for FM generation	**Levels:**	Random
Blocking Variables	Number of Features	**Levels:**	20, 50, 100, 200, 500
	Percentage of Cross-Tree Constraints (with respect to the number of features)	**Levels:**	0%, 5%, 10%, 20%
	Order used when configuring a model	**Levels:**	preorder, inorder, postorder, from more options to less, from less options to more, alphabetically
Constants	SAT solver	**Value:**	Sat4j
	Heuristic for variable selection in the SAT solver	**Value:**	*Default*

Table 13.5: *Hypotheses and design of experiment.*

We can highlight that *Inorder* offers better time consumption results making lnvar more responsive. For example, as shown in Figure §13.4, for models with two hundred features *Inorder* only requires half of the time of all other orderings.

Our conclusion is that the order we select to configure an lnvar model affects the time required by the back-end tool to analyze it. Those results show that the *Inorder* order requires less computational resources than the rest of orders with the models used as input. Therefore, it should be used by default in the scenarios where no specific order is required.

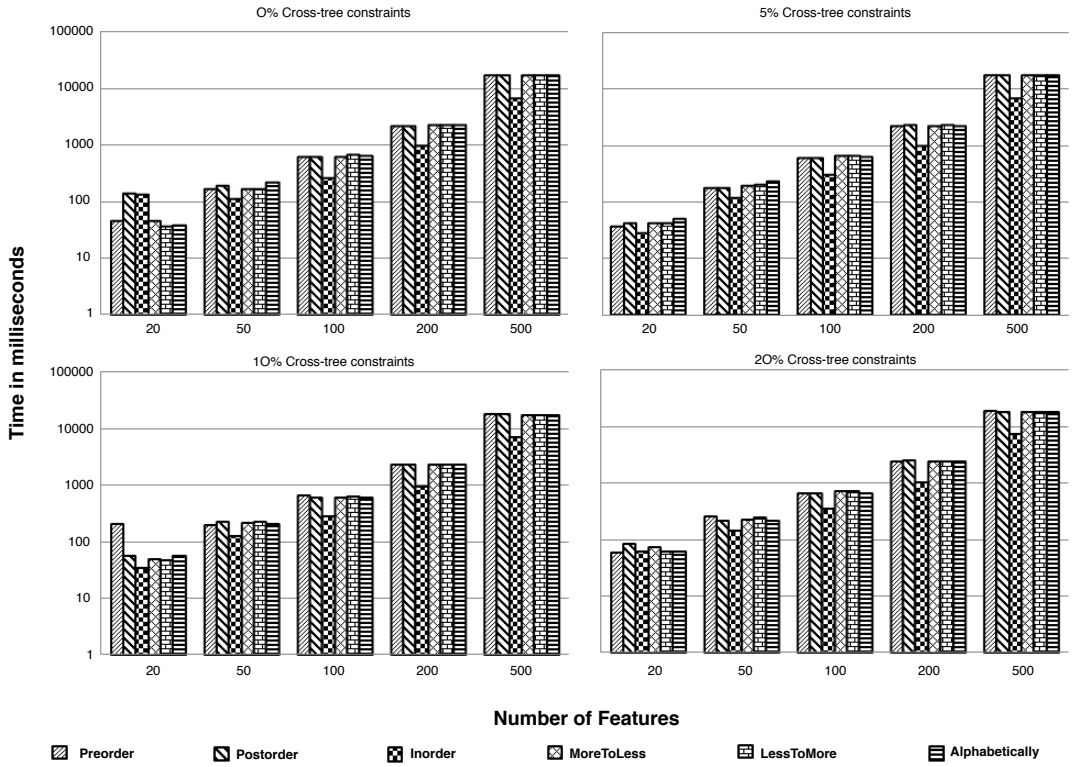


Figure 13.4: *Milliseconds per # features and percentage of constraints.*

13.4 Summary

Variability intensive systems introduced new constraints and contracts for the ussage of variability modelling techniques. Nowadays multiple practitioners configure the same variability intensive systems and different views are used by them to better configure a –mostly distributed– system.

In this chapter, we evaluated the lnvar approach in front of two real scenarios. First, to solve the problem within the Android permission systems (from the variability point of view). Second, in front of a ERP systems widely used in the industry. Moreover, we went throug the different orderings possible when configure a variability model showing that in-order is the easiest in terms of computing capabilities required.

Part VI

Final Remarks

Chapter 14

Conclusions and future work

Una verde, que se ha hecho de la aurora un cinturón, despliega sobre ti un ala de delicia al concederte un espíritu triunfante.

Abu Asbag Ibn Arqam, Andalusian poet

14.1 Conclusions

In this dissertation we have shown that:

Improving the configuration, testing and evolution of variability intensive systems by using the automated analysis of variability models is not only feasible but also improves efficiency and reduces costs.

The automated analysis of feature models have been attracting researchers attention for the last twenty-five years. Now, what was created to only support software product lines have been successfully applied to a variety of contexts. These contexts ranges from cloud systems to software ecosystems such as Android. Moreover, as the research problems drift, new research challenges appears. This motivates the need for new research and new solutions. New research that while grounding in previous research, keeps on improving the previous solutions.

In this thesis we faced some problems derived from the new applications of the automated analysis of feature models to variability intensive systems. Concretely, we have explored the testing and evolution, the product configuration and the multi-model variability analysis areas in this thesis. To show

the validity of our approach we have been extensively evaluated and validated our solutions in different real scenarios coming from our collaboration with different industrial partners. Also, motivating scenarios were crawled from real software ecosystems such as Android.

14.1.1 Discussion and open challenges

In the Chapter §1 we detected and enumerated the research questions we were willing to address in this thesis document. Table §14.1 shows the chapters where we target each question with the contributions we published. Also, in the next paragraph we will go trough all different research questions and explaining how we addressed them.

R1 ***What is the current status of the automated analysis of feature models?*** We performed the mapping study you can find in Chapter §4. In this mapping study we updated the state of the art documented by Benavides et al. [19]

R2 ***What are the current trends in the area?*** We performed key-wording process, analyzing more than 250 abstracts and introduction. From this key-wording process we detected five main trends in the area. Namely, i) Testing and evolution of feature models; ii) Product configuration and derivation; iii)Variability and modeling expressiveness; iv)Multi-model variability analysis; and v) Reverse engineering of feature models.

R3 ***Where does the community publish?*** As part of the mapping study we also categorized and identified the different foras that more attracted researchers from the area of feature model automated analysis attention.

R4 ***How can we improve the testing of variability intensive systems?*** To improve the testing of variability intensive systems we chose to describe the variability contained in a feature model, and add quality attribute information which can be exploited to prune, prioritize and package sets of products to tests.

R5 ***How can we select the most profitable set of products in an variability intensive system?*** We chose the Android ecosystem as a target to validate our approach. We have worked within the ATTACK cloud team to implement the variability aware testing and notice that using market-share data we improve the efficiency of our tests.

R6 ***How can we manage to select a good covering set that optimizes one or multiple stakeholders criteria?*** To implement a solution that copes

with multiple stakeholders criteria, we encoded the variability existing in a feature model and developed a t-wise covering set operation that takes into account quality attributes. For taking into account multiple stakeholders opinion, we implemented an a-priori solution and used a genetic algorithm which proved to be efficient enough for the MOTIV project.

R7 ***How can we grant the safeness when deriving a set of configurations in a variability intensive system that spans over multiple configuration steps?*** The life of a variability intensive system spans over multiple time slots. Usually, companies design the transition in between different time slots. To model and optimize this transition in between configurations we encoded the whole problem as a CSP and validated our approach with our aeronautics industry partners.

R8 ***How can we enable different practitioners to work in tandem over multiple models describing a variability intensive system?*** The large amount of variability encoded in a variability intensive system makes unfeasible to have all the variability encoded in only one model. Also, users might have different privileges so they can only configure some parts. To help in this scenario we developed a software solution than enables the configuration of variability intensive systems using different modeling paradigms such as feature models, orthogonal variability models and dopler models.

Research question	Chapter	Published contributions
R1	§4	–
R2	§4	–
R3	§4	–
R4	§6,§10	[71, 177]
R5	§6,§10	[71, 177]
R6	§7,§11	[2, 7, 69]
R7	§8,§12	[179]
R8	§9,§13	[55, 57]

Table 14.1: *Links between chapters, research questions and publications.*

Then, back to the original question:

How to apply the automated analysis of feature model in variability intensive systems?

By addressing the previous research questions we can now assert that is possible to apply the automated analysis of variability models to variability intensive systems. This also incurs in cost reduction and increased efficiency of software engineering activities such as configuration, testing and evolution.

14.2 Future work

In addition to the future work mentioned in each chapter of this dissertation here we show other future work required to cover all current research trends in the automated analysis of feature models.

Reverse engineering of feature models: Nowadays it is a common practice to offer a system in many variants such as community, professional, or academic editions. Each variant provides different functionality described in terms of features. As practices are becoming more pervasive, reverse engineering feature models from the feature descriptions of each individual variant has become an active research subject. Then, is appealing to apply reverse engineering techniques to variability intensive system. Admittedly we already stared to work in this direction obtaining some publications such as [104, 105].

Error detection and correction in variability intensive systems:Due to the increasing size and complexity of feature models, anomalies in terms of inconsistencies and redundancies can occur which lead to increased efforts related to feature model development and maintenance. We plan to introduce new knowledge representations which serve as a basis for the explanation of anomalies in feature models. Again, we also started to work in this area collaborating with some authors [17, 66].

Scalable analysis of feature models: The large size of models encoding variability intensive systems variability require large computation capabilities. Also, the new constructs enhance the combinatorial explosion which makes unusable variability intensive systems analysis techniques. As part of our future work, we will explore the different ways of better scaling the automated analysis of feature models.

Industrial applications: We think that some of the techniques developed during the writting of this thesis can be applied in the industry. For example, is appealing to develop a cloud system for testing Android applications that enables the optimization of multiple parameters. We currently are in context

with some companies in order to move this idea forward. Also, regarding the video-surveillance testing, we are collaborating with the INPIXAL company so we can offer a platform for better testing those algorithms.

On the needing of new variability constructs: During the elaboration of this thesis we detected that current basic variability mechanisms are useful but not enough, attributes and multi-features are of prior importance, and meta-information is relevant for efficient variability analysis. In addition, we questioned the existence of one-size-fits-all variability modeling solution applicable in any industry. Yet, some common needs for modeling variability are becoming apparent such as support for attributes and multi-features. We already stared to investigate this in the context of video-sequences testing [8]

Part VII

Appendix

Chapter 15

WindRose

El que no llora no mama

Dicho popular, Andalusian people

Feature modelling is the "de facto" standard to describe the common and variant parts of software product lines. Different tools, approaches and operations for the automated analysis of feature models have been proposed in the last 20 years. However, the installation and usage of those tools use to be time consuming. In this chapter we present the WindRose IDE, a cloud based IDE that allows the storage, edition and analysis of feature models while being executed in the cloud. WindRose integrates different feature model analysis operations such as Valid or Number of Products. This reasoning capabilities rely in different well stablished tools like FaMa or FaMiliar. Moreover, WindRose generates a public repository where practitioners can share and distribute their models[†1].

[†1] Part of this material have been published in the JISBD conference[70]

15.1 Introduction

Software product lines (SPL) are about developing a set of different software products that share some common functionality [38]. Feature models (FMs) is the standard to represent the variants and commonalities of a software product line. A feature model can encode a large set of products such as those describing operating systems variability [160] or cloud computing providers [73].

To extract useful information from feature models, researchers and practitioners proposed the use of computer-aided techniques. This is commonly called Automated Analysis of feature models. In the last 20 years of feature modeling, practitioners released different analysis tools and more than 30 operations were proposed in the literature such as "NumberOfProducts" or "ValidModel".

Currently there is a diversity of tools, operations and textual formats. However, the tediousness of configuring each tool hinders the collaboration and reuse of them. In this chapter, we present the WindRose tool. WindRose is a cloud-based IDE that enables the analysis operations execution in the cloud. This easies off the requirement of installing and configuring each tool and enables the collaboration between practitioners when designing a feature model.

The main features of WindRose are:

- Allows the simultaneous editing of feature models in textual language.
- Enables the reasoning of feature models using well-tested tools such as FaMa or FaMiLiAr.
- Offers two repositories, the first public, where all users can share their models, the second private, where researchers can build their models without prying eyes.
- Is open source, this is, practitioners can install it in their own server for private use.

15.2 WindRose cloud-based IDE

WindRose is a cloud-based IDE that integrates different automated reasoning tools. It has been developed on top of the CODIAD IDE (http://codiad.

com/) which has been developed by the MIT and provides useful collaboration tools and a plugin system out of the box. Figure §15.1 shows a brief overview of the architecture. Several parts of CODIAD have been extended, for example, the plugin system which in the case of WindRose use REST web-services to communicate and isolate from the different tool implementations.

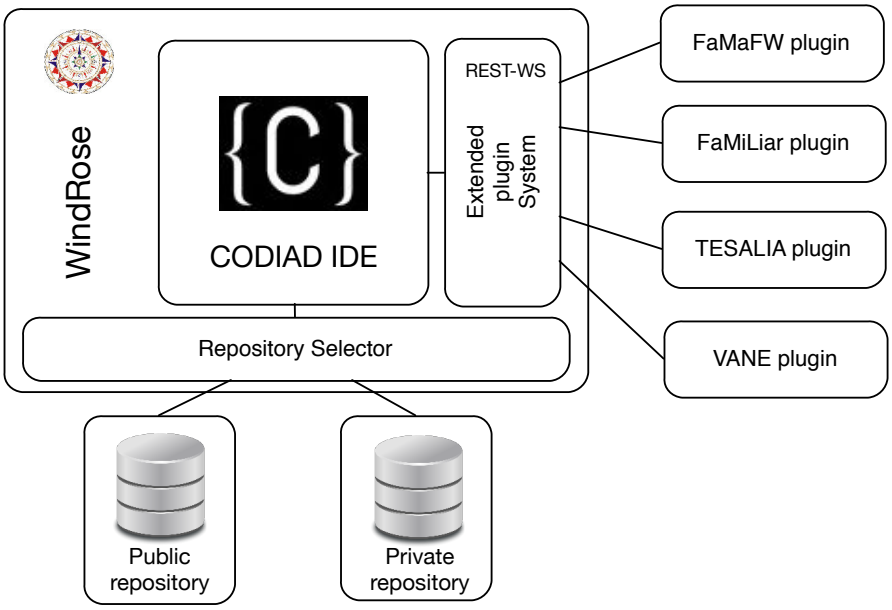


Figure 15.1: *WindRose architecture overview.*

15.2.1 Available plugins

WindRose offers out of the box integration with different analysis tools. In WindRose, each plugin is associate with one textual format. This is used to filter by the file extension the available analysis operations. Some plugin implementations are:

FaMaTS-SAT FaMa [20] is a framework for building variability analysis tools. Different tools have been build on top of it's architecture [68, 145, 156]. It also offers different ways of consume its services: i) by OsGi; i) by Web services (REST and SOAP); iii) by terminal and iv) by importing the libraries in a java program. Concretely we configured this tool to use

the SAT4j [23] based implementation of the analysis operations for traditional feature models. This tool is associated with the files having the .fama extension.

FaMaTS-CSP for Attributed FM FaMa enables the analysis of attributed feature models. Internally this plugin uses the Choco2 solver http://www.emn.fr/z-info/choco-solver/ as back-end for executing the analysis operations. This plugin is associated with the extension .afm and .vm

FaMiLiAr FaMiLiar[4] is a framework for managing families of products while enabling operations such as merging different models. This plugin is associated with the extension .tvl.

VANE This is a proposal developed by INRIA Rennes in France. It allows to obtain the pair-wise coverages that maximizes a custom attribute value [69]. It is associated with the vm [7] file format.

TESALIA This is a proposal for the testing of feature models containing quality information. This tool is also used internally as part of the ATAACK cloud [177] and help with the testing of Android apps. This plugin is associated with the afm file format.

15.2.2 Analysis operations

WindRose offers out of the box different feature model operations. Note that these operation can be extending by implementing new plugins. Currently the supported operations are:

- *Valid Model.* This operations checks the semantic correctness of a feature model. This is, if it represent at least one product. This operation is implemented by the FaMaTS-SAT, FaMaTS-CSP for Attributed FM and FaMiLiAr plugins.

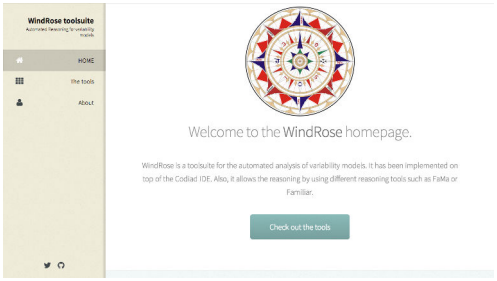
- *Valid Configuration.* This operation takes a model and a partial configuration and check if the configuration is correct or not. This operation is implemented by the FaMaTS-SAT and FaMaTS-CSP for Attributed FM plugins.

- *Valid Product.* This operation takes a product (a.k.a non-partial configuration) and checks its validity on top of the selected model. This operation is implemented by the FaMaTS-SAT and FaMaTS-CSP for Attributed FM plugins.

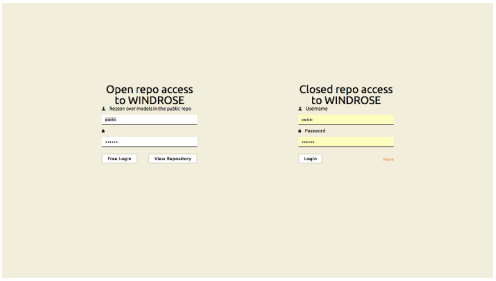
- *All products.* This operation print out the list of valid products from a feature model. FaMaTS-SAT, FaMaTS-CSP for Attributed FM and FaMiLiAr plugins.
- *Dead Features.* This operation detects those features that cannot be selected because of the topology of the model. This operation is implemented by the FaMaTS-SAT and FaMaTS-CSP for Attributed FM plugins.
- *Core Features.* This operation returns the features present in all products. This operation is implemented by the FaMaTS-SAT and FaMaTS-CSP for Attributed FM plugins.
- *Pair-Wise.* This operation returns a random pairwise coverage.This operation is implemented by the VANE plugin.
- *Pair-Wise with attributes.* This operation returns the pair-wise coverage that maximize a concrete attribute value. This operation returns a random pairwise coverage.This operation is implemented by the VANE plugin.
- *Prioritization.* This operation returns the list of products but ordered by a function between attributes. This operation returns a random pairwise coverage. This operation is implemented by the TESALIA plugin.
- *Packaging.* This operation apply a knapsack over products containing value and cost definition. This operation is implemented by the TESALIA plugin.
- *Prunning.* This operation returns the set of products that meet a custom constraint over attributes. This operation is implemented by the TESALIA plugin.

15.2.3 Repositories

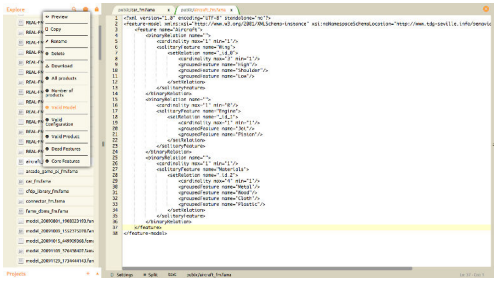
WindRose offers two repositories. In the first hand, a public repository that does not require registration and allows users to download any model stored and execute analysis operations depending on the size of the models. For example, the operation that retrieves all valid configurations from a model is restricted to small-size models in the public repository because of the large computing capabilities required to execute the operation. However, models in the public repository can not be overwritten. In the second hand, registered users can work privately with all-size models. Also these models can be



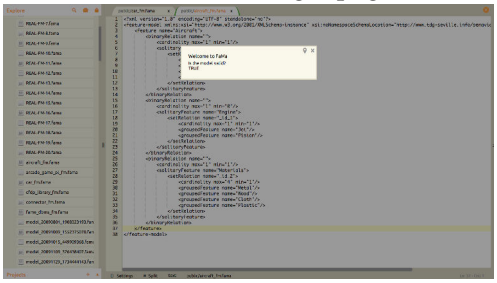
(a) WindRose home page



(b) WindRose login page



(c) WindRose public repository FaMaTS-SAT execution



(d) WindRose public repository FaMaTS-SAT result

Figure 15.2: *Screenshots of WindRose in action.*

moved to the public repository so they can be publicly released. Figure§15.2 shows some screenshots of WindRose public repository.

Another characteristic of the WindRose public repository is that allows researchers to associate the model with a publication. This is to store citation data (in bibtex format) among the model itself.

Chapter 16

Foras where the community publish

In this section we show the different foras we found while doing the mapping study showed in Chapter §4.

- International Symposium on Methodologies for Intelligent Systems (ISMIS)
- International Symposium of Applications of Graph Transformation With Industrial Relevance (AGTIVE)
- International Nordic Conference on Secure IT Systems (NordSec)
- International Conference of Testing Software and Systems (ICTSS)
- International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2013)
- International Workshop on Knowledge Engineering and Software Engineering (KESE)
- International Conference on Integrated Formal Methods (iFM)
- International Conference of Distributed Computing and Internet Technology (ICDCIT)
- The Workshop on Information Systems Economics (WISE)
- International Conference on Model Transformation (ICMT)
- International Conference on Aspect-oriented Software Development (AOSD)
- European Conference on Modeling Foundations and Applications (ECMFA) (International)
- International Symposium on Applied Computing (SAC)
- International Conference on Engineering of Complex Computer Systems (ICECCS)
- International Conference on Software Engineering Research (SERP)
- International Conference on Software Language Engineering (SLE)
- International FME Workshop on Formal Methods in Software Engineering (FORMALISE)
- International Workshop on Variability & Composition (VariComp)
- International Conference on Formal Verification of Object-Oriented Software (FoVeOOS)

- International Conference on Software Testing, Verification and Validation Workshops (ICSTW)
- International Conference on Formal Methods for Components and Objects (FMCO)
- International Comparing *Requirements* Modeling Approaches workshop (CMA@RE)
- International Symposium Meeting on Foundations of Software Engineering (FSE)
- International Conference on Advances in Conceptual Modeling: Recent Developments and New Directions (ER)
- International Requirements Engineering Conference (RE)
- International Configuration Workshop (CW)
- International Workshop on automated configuration and tailoring of applications (ACoTA)
- International Conference on Cloud Computing and Services Science (CLOSER)
- International Workshop on High-Performance Stencil Computations (HiStencils)
- International Conference on Concurrent Engineering (ISPE CE)
- Working Conference on Reverse Engineering (WCRE)
- International Workshop on Advances in Model Based Testing (A-MOST)
- International Symposium on Software Reliability Engineering (ISSRE)
- International Workshop on Configuration Systems (ECAI)
- International Conference on Principles and Practice of Constraint Programming (CP)
- International Conference on Application of Information and Communication Technologies (AICT)
- International Conference on Object oriented programming systems languages and applications (OOPSLA)
- International Conference on Concurrent Engineering (CE)
- International Conference on Information Resources Management (Conf-IRM)
- International Conference on Services Computing (SCC)
- International Conference on Advances in Mobile Computing and Multimedia (MoMM)
- International Conference on Software Maintenance (ICSM)
- Working Conference on Software Visualization (VISSOFT)
- Conferencia Latinoamericana En Informatica (CLEI)
- International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)
- International Workshop on Satisfiability Modulo Theories (SMT)
- Asia-Pacific Software Engineering Conference (APSEC) (International)
- International Conference on Software Engineering and Advanced Applications (SEAA)
- Software Engineering Conference (APSEC) (International)
- International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)

- International symposium on Engineering interactive computing systems (EICS)
- International Workshop on Software Product Management (IWSPM)
- International Workshop on Requirements Engineering Practices on Software Product Line Engineering (REPOS)
- African Conference on Software Engineering and Applied Computing (ACSEAC)
- International Conference on Concurrent Engineering (IPSE)
- First International Workshop on Combining Modeling and Search-Based Software Engineering (CMSBSE)
- International Workshop on Recommendation Systems for Software Engineering (RSSE)
- International Workshop on Replication in Empirical Software Engineering Research (RESER)
- International Workshop on Multi Product Line Engineering (MultiPLE)
- International Conference on Software Testing, Verification and Validation Workshop (ICSTW)
- International Annual Conference on Systems Engineering Research (CSER 2013)
- The Genetic and Evolutionary Computation Conference (GECCO)
- International Conference on Software Quality (ICSQ)
- International Conference on Towards a Service-based Internet (ServiceWave)
- International Workshop on Variability-intensive Systems Testing, Validation, and Verification (VAST)
- International Workshop on Knowledge-Oriented Product Line Engineering (KOPLE)
- International Conference on Software Engineering and Formal Methods (SEFM)

Bibliography

[1] A. Abran and J. Moore. *Guide to the software engineering body of knowledge*. IEEE Computer Society, 2004

[2] M. Acher, M. Alferez, J. A. Galindo, P. Romenteau, and B. Baudry. Vivid: A variability-based tool for synthesizing video sequences. In *18th International Software Product Line Conference (SPLC'14), tool track*, Florence, Italie, 2014.

[3] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 45–54, New York, NY, USA, 2012. ACM

[4] M. Acher, P. Collet, P. Lahire, and R. France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming (SCP) Special issue on programming languages*, page 22, 2013

[5] M. Acher, P. Heymans, P. Collet, C. Quinton, P. Lahire, and P. Merle. Feature model differences. In *International Conference on Advanced Information Systems Engineering (CAISE)*, LNCS. Springer, jun 2012.

[6] M. Akbar, E. Manning, G. Shoja, and S. Khan. Heuristic solutions for the multiple-choice multi-dimension knapsack problem. *Computational Science-ICCS 2001*, pages 659–668, 2001

[7] M. Alférez, J. A. Galindo, M. Acher, and B. Baudry. Modeling Variability in the Video Domain: Language and Experience Report. Rapport de recherche RR-8576, INRIA, July 2014.

[8] M. Alférez, J. A. Galindo, M. Acher, B. Baudry, and D. Benavides. Modeling variability in the video domain: Language and experience report. *Information and Software Technology Journal*, 2015

[9] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines*. Springer, 2013.

[10] K. Bak, K. Czarnecki, and A. Wasowski. Feature and meta-models in clafer: Mixed, specialized, and coupled. In *International Conference on Software Language Engineering (SLE)*, SLE'10, pages 102–122, Berlin, Heidelberg, 2011. Springer-Verlag.

[11] D. Batory. Feature-oriented programming and the ahead tool suite. In *Proceedings of the 26th International Conference on Software Engineering*, pages 702–703. IEEE Computer Society, 2004

[12] D. Batory. Feature models, grammars, and propositional formulas. *Software Product Lines*, pages 7–20, 2005

[13] D. Batory, D. Benavides, and A. Ruiz-Cortes. Automated analysis of feature models: challenges ahead. *Communications of the ACM*, 49(12): 45–47, 2006

[14] K. Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003

[15] B. Beizer. *Software testing techniques*. Dreamtech Press, 2003

[16] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In *Advanced Information Systems Engineering*, pages 381–390. Springer, 2005

[17] D. Benavides, A. Felfernig, J. A. Galindo, and F. Reinfrank. Automated analysis in feature modelling and product configuration. In *International Conference on Software Reuse (ICSR)*, pages 160–175, 2013.

[18] D. Benavides and J. G. Galindo. Variability management in an unaware software product line company: An experience report. In *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*, VaMoS '14, pages 5:1–5:6, New York, NY, USA, 2013. ACM

[19] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: a literature review. *Information Systems*, 35(6), 2010

[20] D. Benavides, P. Trinidad, A. R. Cortés, and S. Segura. *FaMa*, chapter FaMa, pages 163–171. Springer Berlin Heidelberg, 2013

[21] D. Benavides, P. Trinidad, A. Ruiz-Cortés, and S. Segura. Fama. In *Systems and Software Variability Management*, pages 163–171. Springer, 2013

[22] Berger, Thorsten and Rublack, Ralf and Nair, Divya and Atlee, Joanne M. and Becker, Martin and Czarnecki, Krzysztof and Wasowski, Andrzej. A survey of variability modeling in industrial practice. In *VaMoS'13*. ACM, 2013

[23] D. L. Berre and A. Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3): 59–6, 2010

[24] A. Bertolino and S. Gnesi. Use case-based testing of product lines. *ACM SIGSOFT Software Engineering Notes*, 28(5):355–358, 2003

[25] D. Beuche. Variant management with pure:: variants. *Pure-systems GmbH, Tech. Rep*, 2003

[26] D. Binkley and I. C. Society. Test Cost Reduction. 23(8):498–516, 1997

[27] B. Boehm. Value-Based Software Engineering : Seven Key Elements and Ethical Considerations. (February), 2005

[28] B. Boehm and K. Sullivan. Software Economics : A Roadmap. pages 319–343, 1992

[29] J. Bosch. From software product lines to software ecosystems. In *SPLC 2009*, pages 111–119, San Francisco, CA, USA, 2009. ACM ICPS

[30] J. Bosch. Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization. pages 257–271, August 2002.

[31] G. Botterweck, A. Pleuss, A. Polzer, and S. Kowalewski. Towards feature-driven planning of product-line evolution. In *Proceedings of the First International Workshop on Feature-Oriented Software Development*, pages 109–116. ACM, 2009

[32] G. Botterweck, M. Janota, and D. Schneeweiss. A design of a configurable feature model configurator. In *3rd International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2009)*, pages 165–168, Sevilla, Spain, 2009. ICB Research Report vol. 29

[33] R. Buhrdorf, D. Churchett, and C. Krueger. Salion's experience with a reactive software product line approach. *Software Product-Family Engineering*, pages 317–322, 2004

[34] J. Burns. Developing secure mobile applications for android, 2008

[35] L. Chen, M. Babar, and N. Ali. Variability management in software product lines: A systematic review. In *SPLC 2009*, pages 81–90, San Francisco, CA, USA, 2009. ACM ICPS

[36] A. Classen, A. Hubaux, and P. Heymans. A formal semantics for multi-level staged configuration. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 51–60, 2009

[37] A. Classen, Q. Boucher, and P. Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Science of Computer Programming*, 76(12):1130 – 1143, 2011

[38] P. Clements and L. Northrop. *Software product lines*. Addison-Wesley Boston, 2002

[39] C. A. Coello Coello. Evolutionary multi-objective optimization: a historical view of the field. *Computational Intelligence Magazine, IEEE*, 1 (1):28–36, 2006

[40] E. Coffman Jr, M. Garey, and D. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996

[41] M. B. Cohen, M. B. Dwyer, and J. Shi. Coverage and adequacy in software product line testing. *Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis - ROSATEA '06*, pages 53–63, 2006

[42] M. B. Cohen, M. B. Dwyer, and I. C. Society. Constructing Interaction Test Suites for Highly-Configurable Systems in the Presence of Constraints : A Greedy Approach. 34(5):633–650, 2008

[43] T. E. Colanzi, W. K. G. Assunção, D. de Freitas Guilhermino Trindade, C. A. Zorzo, and S. R. Vergilio. Evaluating Different Strategies for Testing Software Product Lines. *Journal of Electronic Testing*, 29(1):9–24, February 2013

[44] B. Combemale, O. Barais, O. Alam, and J. Kienzle. Using cvl to operationalize product line development with reusable aspect models. In *VARiability for You Workshop: Variability Modeling Made Useful for Everyone (VARY)*, VARY '12, pages 9–14, New York, NY, USA, 2012. ACM.

[45] S. P. Consortium. Synthesis guidebook. Technical report, SPC-91122-MC. Herndon, Virginia: Software Productivity Consortium, 1991

[46] M. Cordy, P.-Y. Schobbens, P. Heymans, and A. Legay. Beyond boolean product-line model checking: Dealing with feature attributes and multi-features. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 472–481, May 2013

[47] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. *Software Product Lines Conference*, pages 162–164, 2004

[48] K. Czarnecki and A. Wasowski. Feature diagrams and logics: There and back again. In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 23–34, 2007

[49] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *PROC. INTL. CONF. ON SOFTWARE ENGINEERING (ICSE '99)*, pages 285–294, 1999

[50] S. M. Davis. Future perfect. 1987

[51] K. Deb. Multi-objective optimization. *Multi-objective optimization using evolutionary algorithms*, pages 13–46, 2001

[52] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002

[53] G. Deng, D. C. Schmidt, A. Gokhale, J. Gray, Y. Lin, and G. Lenz. Evolution in model-driven software product-line architectures. *Designing Software-Intensive Systems: Methods and Principles*, 2008

[54] D. Dhungana, P. Grünbacher, and R. Rabiser. The DOPLER meta-tool for decision-oriented variability modeling: A multiple case study. *Automated Software Engineering*, 18(1):77–114, 2011

[55] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Grünbacher, D. Benavides, and J. Galindo. Integrating heterogeneous variability modeling approaches with invar. In *7th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS 2013)*, Pisa, Italy, 2013. ACM

[56] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Grünbacher, D. Benavides, and J. Galindo. Configuration of multi product lines by

bridging heterogeneous variability modeling approaches. In *International Software Product Line Conference (SPLC)*, pages 120–129, 2011.

[57] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Grünbacher, D. Benavides, and J. Galindo. Configuration of multi product lines by bridging heterogeneous variability modeling approaches. In E. S. de Almeida, T. Kishi, C. Schwanninger, I. John, and K. Schmid, editors, *SPLC*, pages 120–129. IEEE, 2011

[58] I. do Carmo Machado, J. D. McGregor, and E. Santana de Almeida. Strategies for testing products in software product lines. *ACM SIGSOFT Software Engineering Notes*, 37(6):1, November 2012

[59] B. Dougherty, J. White, R. Kegley, J. Preston, D. Schmidt, and A. Gokhale. Optimizing integrated application performance with cache-aware metascheduling. *On the Move to Meaningful Internet Systems: OTM 2011*, pages 432–450, 2011

[60] B. Dougherty, J. White, and D. C. Schmidt. Model-driven auto-scaling of green cloud computing infrastructure. *Future Generation Comp. Syst.*, 28(2):371–378, 2012

[61] E. Dustin, J. Rashka, and J. Paul. *Automated software testing: introduction, management, and performance*. Addison-Wesley Professional, 1999

[62] C. Elsner, G. Botterweck, D. Lohmann, and W. Schroder-Preikschat. Variability in time—product line variability and evolution revisited. 2010

[63] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of android application security. In *USENIX security symposium*, volume 2, page 2, 2011

[64] W. Enck, M. Ongtang, P. D. McDaniel, and others. Understanding android security. *IEEE security & privacy*, 7(1):50–57, 2009

[65] L. Etxeberria and G. Sagardui. Variability driven quality evaluation in software product lines. In *Software Product Line Conference, 2008. SPLC'08. 12th International*, pages 243–252. IEEE, 2008

[66] A. Felfernig, D. Benavides, J. Galindo, and F. Reinfrank. Towards anomaly explanations in feature models. In *International Configuration Workshop (CW)*, pages 117–124, 2013.

[67] G. Fleischanderl, G. E. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner. Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems*, 13(4):59–68, July 1998

[68] J. Galindo, D. Benavides, and S. Segura. Debian packages repositories as software product line models. towards automated analysis. In *ACoTA*, pages 29–34, 2010

[69] J. A. Galindo, M. Alférez, M. Acher, B. Baudry, and D. Benavides. A variability-based testing approach for synthesizing video sequences. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 293–303, New York, NY, USA, 2014. ACM

[70] J. A. Galindo, D. Benavides, M. Alférez, M. Acher, and B. Baudry. WindRose: A Cloud-Based Integrated Development Environment for the Automated Analysis of Feature Models. In *Jornadas de Ingeniería del Software y Bases de Datos (JISBD) Sistedes;2014.*, 2014

[71] J. A. Galindo, H. Turner, D. Benavides, and J. White. Testing variability intensive systems using automated analysis. an application in android. *Software Quality Journal*, 2014

[72] J. A. Galindo, D. Dhungana, G. Botterweck, R. Rabiser, P. Grünbacher, and D. Benavides. Supporting distributed product configuration by integrating heterogeneous variability modeling approaches. *Information and Software Technology*, 2014

[73] J. García-Galán, O. F. Rana, P. Trinidad, and A. Ruiz-Cortés. Migrating to the cloud: a software product line based analysis. In *3rd International Conference on Cloud Computing and Services Science (CLOSER'13)*, 2013

[74] J. García-Galán, P. Trinidad, J. A.Galindo, and A. Ruiz-Cortés. Tool supported error detection and explanations on feature models. In *Proc. of 2nd International Workshop on Formal Methods and Analysis in Software Product Line Engineering (FMSPLE 2011), co-located with Software Product Line Conference 2011 (SPLC 2011)*, pages 6–6, Munich, 08/2011 2011. Fraunhofer, Fraunhofer

[75] C. D. Gill, J. P. Loyall, R. E. Schantz, M. Atighetch, J. M. Gossett, D. Gorman, and D. C. Schmidt. Integrated adaptive qos management in middleware: A case study. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pages 276–285. IEEE, 2004

[76] A. Hartman. Software and hardware testing using combinatorial covering suites. In *Graph Theory, Combinatorics and Algorithms*, pages 237–266. Springer, 2005

[77] H. Hartmann, T. Trew, and A. Matsinger. Supplier independent feature modelling. In *Proceedings of the 13th International Software Product Line Conference*, pages 191–200. Carnegie Mellon University, 2009

[78] E. Haslinger, R. Lopez-Herrejon, and A. Egyed. Reverse engineering feature models from programs' feature sets. In *Reverse Engineering (WCRE), 2011 18th Working Conference on*, pages 308–312, 2011

[79] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le Traon. Towards automated testing and fixing of re-engineered feature models. In *International Conference on Software Engineering (ICSE)*, pages 1245–1248, 2013.

[80] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. L. Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test suites for large software product lines. *IEEE Transactions on Software Engineering*, 2014.

[81] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon. Multi-objective test generation for software product lines. In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, pages 62–71, New York, NY, USA, 2013. ACM

[82] A. Hubaux, A. Classen, and P. Heymans. Formal modelling of feature configuration workflows. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 221–230, Pittsburgh, PA, USA, 2009. Carnegie Mellon University

[83] C. Hwan, P. Kim, and K. Czarnecki. Synchronizing cardinality-based feature models and their specializations. In *Model Driven Architecture–Foundations and Applications*, pages 331–348. Springer, 2005

[84] S. Ida and S. Ketil. Technology research explained. Technical report, 2007

[85] A. Immonen. A method for predicting reliability and availability at the architectural level. *Software Product Lines*, pages 373–422, 2006

[86] M. F. Johansen, O. y. Haugen, and F. Fleurey. An algorithm for generating t-wise covering arrays from large feature models. *Proceedings of the 16th International Software Product Line Conference on - SPLC '12 -volume 1*, page 46, 2012

[87] M. F. Johansen, O. Haugen, F. Fleurey, A. G. Eldegard, and T. Syversen. Generating better partial covering arrays by modeling weights on sub-product lines. In *Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems*, MODELS'12, pages 269–284, Berlin, Heidelberg, 2012. Springer-Verlag

[88] José A.Galindo, F. Roos-Frantz, J. García-Galán, and A. Ruiz-Cortés. Extracting orthogonal variability models from debian repositories. In *Proc. of 2nd International Workshop on Formal Methods and Analysis in Software Product Line Engineering (FMSPLE 2011), co-located with Software Product Line Conference 2011 (SPLC 2011)*, pages 8–8, Munich, 08/2011 2011. Fraunhofer, Fraunhofer

[89] W. E. Juengst and M. Heinrich. Using resource balancing to configure modular systems. *IEEE Intelligent Systems*, 13(4):50–58, July 1998

[90] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Technical Report CMU/SEI-90TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 1990

[91] K. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. Form: A feature-; oriented reuse method with domain-; specific reference architectures. *Annals of Software Engineering*, 5(1):143–168, 1998

[92] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, DTIC Document, 1990

[93] C. Kästner, A. von Rhein, S. Erdweg, J. Pusch, S. Apel, T. Rendel, and K. Ostermann. Toward variability-aware testing. In *Proceedings of the 4th International Workshop on Feature-Oriented Software Development*, FOSD '12, pages 1–8, New York, NY, USA, 2012. ACM

[94] D. Kuhn, R. Kacker, and Y. Lei. Practical combinatorial testing. *NIST Special Publication*, 800:142, 2010

[95] D. Kuhn, D. Wallace, and J. Gallo, A.M. Software fault interactions and implications for software testing. *Software Engineering, IEEE Transactions on*, 30(6):418–421, 2004

[96] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1):32, 1992

[97] I. Kumara, J. Han, A. Colman, T. Nguyen, and M. Kapuruge. Sharing with a difference: Realizing service-based saas applications with runtime sharing and variation in dynamic software product lines. In *International Conference on Services Computing (SCC)*, pages 567–574, 2013.

[98] B. P. Lamancha and M. P. Usaola. Testing product generation in software product lines using pairwise for features coverage. In *Proceedings of the 22Nd IFIP WG 6.1 International Conference on Testing Software and Systems*, ICTSS'10, pages 111–125, Berlin, Heidelberg, 2010. Springer-Verlag

[99] M. M. Lehman. Laws of Software Evolution Revisited. 1968.

[100] M. M. Lehman. Program evolution. *Information Processing & Managment*, 20(1), 1984.

[101] Y. Lei and K.-C. Tai. In-parameter-order: A test generation strategy for pairwise testing. In *High-Assurance Systems Engineering Symposium, 1998. Proceedings. Third IEEE International*, pages 254–261. IEEE, 1998

[102] F. Liguori and F. Schreiber. The software configurator : an aid to the industrial production of software. In *Computer Software and Applications Conference, 1978. COMPSAC '78. The IEEE Computer Society's Second International*, pages 487–492, 1978

[103] L. B. Lisboa, V. C. Garcia, D. L. dio, E. S. de Almeida, S. R. de Lemos Meira, and R. P. de Mattos Fortes. A systematic review of domain analysis tools. *Information and Software Technology*, 52(1):1–13, 2010

[104] R. E. Lopez-Herrejon, L. Linsbauer, J. A. Galindo, J. A. Parejo, D. Benavides, S. Segura, and A. Egyed. An assessment of search-based techniques for reverse engineering feature models. *Journal of Systems and Software*, 2014

[105] R. E. Lopez-Herrejon, J. A. Galindo, D. Benavides, S. Segura, and A. Egyed. Reverse engineering feature models with evolutionary algorithms: An exploratory study. In *4th Symposium on Search Based Software Engineering*, pages 168–182, Trento, Italy, 2012. Springer

[106] R. Lotufo, S. She, T. Berger, K. Czarnecki, and A. Wasowski. Evolution of the linux kernel variability model. *Software Product Lines: Going Beyond*, pages 136–150, 2010

[107] G. Madl, S. Abdelwahed, and D. C. Schmidt. Verifying distributed real-time properties of embedded systems via graph transformations and model checking. *Real-Time Systems*, 33(1):77–100, 2006

[108] M. Mannion. Using first-order logic for product line model validation. *Software Product Lines*, pages 149–202, 2002

[109] D. Marijan, A. Gotlieb, S. Sen, and A. Hervieu. Practical pairwise testing for software product lines. In T. Kishi, S. Jarzabek, and S. Gnesi, editors, *SPLC*, pages 227–235. ACM, 2013

[110] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990

[111] M. Mendonca, A. Wasowski, K. Czarnecki, and D. Cowan. Efficient compilation techniques for large scale feature models. In *Proceedings of the 7th international conference on Generative programming and component engineering*, pages 13–22. ACM, 2008

[112] M. Mendonca, M. Branco, and D. Cowan. S.p.l.o.t.: Software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 761–762, New York, NY, USA, 2009. ACM

[113] A. Metzger, P. Heymans, K. Pohl, 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 International Requirements Engineering Conference (RE'07)*, pages 243–253, New Delhi, India, 2007. IEEE CS

[114] S. Mittal and F. Frayman. Towards a generic model of configuraton tasks. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'89, pages 1395–1401, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[115] S. Mohalik, S. Ramesh, J.-V. Millo, S. Krishna, and G. Narwane. Tracing spls precisely and efficiently. In *International Software Product Line Conference (SPLC)*, volume 1, pages 186–195, 2012.

[116] H. Muccini and A. Van Der Hoek. Towards testing product line architectures. *Electronic Notes in Theoretical Computer Science*, 82(6):99–109, 2003

[117] J. Musa. Software-reliability-engineered testing practice (tutorial). In *Proceedings of the 19th international conference on Software engineering*, pages 628–629. ACM, 1997

[118] G. K. Narwane, J. A. Galindo, S. N. Krishna, D. Benavides, J.-V. Millo, and S. Ramesh. Traceability analyses between features and assets in software product lines. *Journal of Logical and Algebraic Methods in Programming*, 2014

[119] A. Nechypurenko, E. Wuchner, J. White, and D. C. Schmidt. Application of aspect-based modeling and weaving for complexity reduction in the development of automotive distributed realtime embedded system. In *Proceedings of the Sixth International Conference on Aspect-Oriented Software Development*, 2007

[120] T. Nguyen, A. Colman, and J. Han. Modeling and managing variability in process-based service compositions. In *International Conference on Service-Oriented Computing (ICSOC)*, ICSOC'11, pages 404–420, Berlin, Heidelberg, 2011. Springer-Verlag.

[121] C. Nie and H. Leung. A survey of combinatorial testing. *ACM Computing Surveys (CSUR)*, 43(2):11, 2011

[122] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. K. Aggarwal, H. Lee, L. Davis, E. Swears, X. Wang, Q. Ji, K. Reddy, M. Shah, C. Vondrick, H. Pirsiavash, D. Ramanan, J. Yuen, A. Torralba, B. Song, A. Fong, A. Roy-Chowdhury, and M. Desai. A large-scale benchmark dataset for event recognition in surveillance video. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 3153–3160, Washington, DC, USA, 2011. IEEE Computer Society

[123] R. Olaechea, S. Stewart, K. Czarnecki, and D. Rayside. Modelling and multi-objective optimization of quality attributes in variability-rich software. In *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages*, NFPinDSML '12, pages 2:1–2:6, New York, NY, USA, 2012. ACM

[124] F. Olumofin and V. Misic. Extending the atam architecture evaluation to product line architectures. In *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, pages 45–56. IEEE, 2005

[125] S. Oster, F. Markert, and P. Ritter. Automated Incremental Pairwise Testing of Software Product Lines. pages 196–210, 2010

[126] J. R. Parker. *Algorithms for image processing and computer vision*. Wiley. com, 2010

[127] G. Pascual, M. Pinto, and L. Fuentes. Run-time adaptation of mobile applications using genetic algorithms. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 73–82, 2013.

[128] L. Passos, M. Novakovic, Y. Xiong, T. Berger, K. Czarnecki, and A. Wasowski. A study of non-boolean constraints in variability models of an embedded operating system. Munich, Germany, 08/2011 2011. ACM.

[129] X. Peng, Y. Yu, and W. Zhao. Analyzing evolution of variability in a software product line : From contexts and requirements to features. *Information and Software Technology*, 53(7):707–721, 2011.

[130] G. Perrouin, S. Oster, S. Sen, J. Klein, B. Baudry, and Y. Traon. Pairwise testing for software product lines: comparison of two approaches. *Software Quality Journal*, pages 605–643, August 2011

[131] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. L. Traon. Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines. *2010 Third International Conference on Software Testing, Verification and Validation*, pages 459–468, 2010

[132] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swinton, UK, UK, 2008. British Computer Society

[133] A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski. Model-driven support for product line evolution on feature level. *Journal of Systems and Software*, 2011

[134] A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski. Model-driven support for product line evolution on feature level. *The Journal of Systems & Software*, 85(10):2261–2274, 2012.

[135] K. Pohl, G. Bockle, and F. Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer-Verlag New York Inc, 2005

[136] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005

[137] K. Pohl and A. Metzger. Software product line testing. *Commun. ACM*, 49(12):78–81, December 2006

[138] J. Ponce, D. Forsyth, E.-p. Willow, S. Antipolis-Méditerranée, R. d'activité RAweb, L. Inria, and I. Alumni. Computer vision: a modern approach. *Computer*, 16:11, 2011

[139] R. Rabiser, P. Grünbacher, and D. Dhungana. Requirements for product derivation support: Results from a systematic literature review and an expert survey. *Information and Software Technology*, 52(3):324–346, 2010

[140] R. Rabiser, P. Grünbacher, and M. Lehofer. A qualitative study on user guidance capabilities in product configuration tools. In *27th IEEE/ACM International Conference Automated Software Engineering (ASE'12)*, Essen, Germany, 2012. ACM

[141] R. Rabiser, D. Dhungana, W. Heider, and P. Grünbacher. Flexibility and end-user support in model-based product line tools. In *Euromicro Conference on Software Engineering and Advanced Applications*, pages 508–511, Patras, Greece, 2009. IEEE CS

[142] R. Rabiser, P. Grünbacher, and D. Dhungana. Supporting product derivation by adapting and augmenting variability models. In *SPLC 2007*, pages 141–150, Kyoto, Japan, 2007. IEEE CS

[143] F. Roos-Frantz, J. Galindo, D. Benavides, and A. Ruiz-Cortés. Fama-ovm: A tool for the automated analysis of ovms. In *International Software Product Line Conference (SPLC)*, volume 2, pages 250–254, 2012.

[144] F. Roos-Frantz, J. A.Galindo, D. Benavides, A. R. Cortés, and J. García-Galán. Automated Analysis of Diverse Variability Models with Tool Support. In *Jornadas de Ingeniería del Software y Bases de Datos (JISBD) Sistedes;2014.*, 2014

[145] F. Roos-Frantz, D. Benavides, A. Ruiz-Cortés, A. Heuer, and K. Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal*, 20(3-4):519–565, 2012

[146] F. Roos-Frantz, J. A. Galindo, D. Benavides, and A. Ruiz-Cortés. Fama-ovm: a tool for the automated analysis of ovms. In *Proceedings of the 16th International Software Product Line Conference-Volume 2*, pages 250–254. ACM, 2012

[147] M. Rosenmüller, N. Siegmund, M. Pukall, and S. Apel. Tailoring dynamic software product lines. *SIGPLAN Not.*, 47(3):3–12, October 2011

[148] G. Rothermel and D. Hall. A Safe , Efcient Regression Test Selection Technique. (2):1–35, 1997

[149] D. Sabin and R. Weigel. Product configuration frameworks-a survey. *Intelligent Systems and their Applications, IEEE*, 13(4):42–49, Jul 1998

[150] N. Sannier, M. Acher, and B. Baudry. From comparison matrix to variability model: The wikipedia case study. In *International Conference on Automated Software Engineering (ASE)*, pages 580–585, 2013.

[151] A. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable product line configuration: A straw to break the camel's back. In *International Conference on Automated Software Engineering (ASE)*, pages 465–474, 2013.

[152] A. Sayyad, T. Menzies, and H. Ammar. On the value of user preferences in search-based software engineering: A case study in software product lines. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 492–501, May 2013

[153] K. Schmid, R. Rabiser, and P. Grünbacher. A comparison of decision modeling approaches in product lines. In *5th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2011)*, pages 119–126, Namur, Belgium, 2011. ACM ICPS

[154] J. Schroeter, S. Cech, S. Götz, C. Wilke, and U. Aßmann. Towards modeling a variable architecture for multi-tenant saas-applications. In *International Workshop on Variability Modeling of Software-Intensive Systems (VAMOS)*, pages 111–120, 2012.

[155] S. Segura, D. Benavides, and A. Ruiz-Cortés ands. Functional testing of feature model analysis tools: a test suite. *Software, IET*, 5(1):70 –82, 02 2011

[156] S. Segura, J. Galindo, D. Benavides, J. Parejo, and A. Ruiz-Cortés. Betty: Benchmarking and testing on the automated analysis of feature models. In U. Eisenecker, S. Apel, and S. Gnesi, editors, *Sixth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'12)*, pages 63–71, Leipzig, Germany, 2012. ACM

[157] S. Segura, J. Galindo, D. Benavides, J. Parejo, and A. Ruiz-Cortés. Betty: Un framework de pruebas para el análisis automático de modelos de características. In *XVII Jornadas de Ingeniería del Software y Bases de Datos*, Almeria, Spain, 2012.

[158] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortés. Automated metamorphic testing on the analyses of feature models. *Information and Software Technology*, 53(3):245–258, 2011

[159] R. Shaw. *Boeing 737-300 to 800*. Zenith Press, 1999

[160] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki. The variability model of the linux kernel. *VaMoS*, 10:45–51, 2010

[161] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki. Reverse engineering feature models. In *ICSE*, pages 461–470, 2011

[162] S. She, U. Ryssel, N. Andersen, A. Wasowski, and K. Czarnecki. Efficient synthesis of feature models. *Information and Software Technology*, (0), 2014

[163] M. Sinnema and S. Deelstra. Classifying variability modeling techniques. *Information and Software Technology*, 49(7):717–739, 2007

[164] B. Smith and M. S. Feather. Challenges and methods in testing the remote agent planner. In *In Proc. 5th Int.nl Conf. on Artificial Intelligence Planning and Scheduling (AIPS*, pages 254–263, 2000

[165] H. Sneed. Value Driven Testing. *2009 Testing: Academic and Industrial Conference - Practice and Research Techniques*, pages 157–166, 2009

[166] A. Spillner, T. Linz, and H. Schaefer. *Software testing foundations: a study guide for the certified tester exam*. O'Reilly Media, Inc., 2011

[167] H. Srikanth, L. Williams, and J. Osborne. System test case prioritization of new and regression test cases. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, pages 10 pp.–, 2005

[168] M. Svahnberg and J. Bosch. Evolution in software product lines: two cases. *Journal of Software Maintenance: Research and Practice*, 11(6): 391–422, November 1999.

[169] A. Svendsen, X. Zhang, R. Lind-Tviberg, F. Fleurey, Ø. Haugen, B. Møller-Pedersen, and G. K. Olsen. Developing a software product line for train control: A case study of cvl. In J. Bosch and J. Lee, editors, *SPLC*, volume 6287 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2010

[170] A. Svendsen, X. Zhang, R. Lind-Tviberg, F. Fleurey, Ø. Haugen, B. Møller-Pedersen, and G. K. Olsen. Developing a software product line for train control: A case study of CVL. In *SPLC 2010*, pages 106–120, Jeju Island, Korea, 2010. Springer

[171] A. Sánchez, S. Segura, and A. Ruiz-Cortés. The drupal framework: A case study to evaluate variability testing techniques. In *International Workshop on Variability Modeling of Software-Intensive Systems (VAMOS)*, 2014.

[172] K.-C. Tai and Y. Lei. A test generation strategy for pairwise testing. *Software Engineering, IEEE Transactions on*, 28(1):109–111, 2002

[173] Q. Y. Tang, P. Friedberg, G. Cheng, and C. J. Spanos. Circuit size optimization with multiple sources of variation and position dependant correlation. In *Advanced Lithography*, pages 65210P–65210P. International Society for Optics and Photonics, 2007

[174] T. Thum, D. Batory, and C. Kastner. Reasoning about edits to feature models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 254–264, Washington, DC, USA, 2009. IEEE Computer Society

[175] P. Trinidad. *Automating the Analysis of Stateful Feature Models*. PhD thesis, University of Seville, http://www.lsi.us.es/~trinidad/docs/tesis.pdf, 2012

[176] M. M. Tseng, J. Jiao, and M. E. Merchant. Design for mass customization. *CIRP Annals-Manufacturing Technology*, 45(1):153–156, 1996

[177] H. Turner, J. White, J. Reed, J. A. Galindo, A. Porter, M. Marathe, A. Vullikanti, and A. Gokhale. *Building a Cloud-based Mobile Application Testbed*. IGI Global, 701 E. Chocolate Avenue Hershey PA 17033-1240, USA, 2012

[178] P. Van Hentenryck. *Constraint satisfaction in logic programming*. 1989

[179] J. White, J. Galindo, T. Saxena, B. Dougherty, D. Benavides, and D. Schmidt. Evolving feature model configurations in software product lines. *Journal of Systems and Software*, 87(1):119–136, 2014.

[180] J. White, D. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated diagnosis of product-line configuration errors in feature models. In *Software Product Line Conference, 2008. SPLC'08. 12th International*, pages 225–234. IEEE, 2008

[181] J. White, D. C. Schmidt, A. Nechypurenko, and E. Wuchner. Model intelligence: an approach to modeling guidance. *UPGRADE Journal*, 9(2): 22–28, April 2008

[182] J. White, D. C. Schmidt, E. Wuchner, and A. Nechypurenko. Automating product-line variant selection for mobile devices. In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 129–140. IEEE, 2007

[183] J. White, D. C. Schmidt, E. Wuchner, and A. Nechypurenko. Automatically composing reusable software components for mobile devices. *Journal of the Brazilian Computer Society*, 14(1):25–44, 2008

[184] R. Wieringa, N. Maiden, N. Mead, and C. Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, 11(1):102–107, 2006

[185] J. Withey. Investment Analysis of Software Assets for Product Lines. 1996

[186] Y. Wu, X. Peng, and W. Zhao. Architecture Evolution in Software Product Line : An Industrial Case Study. *Development*, pages 1–16, 2011.

[187] Y. Xiong, A. Hubaux, S. She, and K. Czarnecki. Generating range fixes for software configuration. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 58–68, Piscataway, NJ, USA, 2012. IEEE Press

[188] H. Zhang, J. E. Fritts, and S. A. Goldman. Image segmentation evaluation: A survey of unsupervised methods. *Computer Vision and Image Understanding*, 110(2):260–280, 2008

[189] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Computing Surveys (CSUR)*, 29(4):366–427, 1997

[190] T. Ziadi, C. Henard, M. Papadakis, M. Ziane, and Y. L. Traon. Towards a language-independent approach for reverse-engineering of software product lines. In *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, pages 1064–1071, 2014

[191] I. Zualkernan. A feature modelling framework for ubiquitous embodied learning games. *Frontiers in Artificial Intelligence and Applications*, 231:198–216, 2011.

These are the publications that are directly related to the thesis document.

Chapter 6 *SQJ'14:* **José A.Galindo**, Hamilton Turner, David Benavides and Jules White. Testing variability-intensive systems using automated analysis. An application to Android. Software Quality Journal 2014. DOI: 10.1007/s11219-014-9258-y (In press). **JCR Impact Factor 0.85**.

Chapter 6 *Book chapter:* H Turner, J White, J Reed, **José A. Galindo**, A Porter, M Marathe, A Vullikanti, A Gokhale. Building a Cloud-based Mobile Application Testbed.Software Testing in the Cloud: Perspectives on an Emerging Discipline, IGI Global, 2013, 382-403, doi:10.4018/978-1-4666-2536-5.ch018.

Chapter 7 *ISSTA'14:* **José A Galindo**, Mauricio Alférez, Mathieu Acher, Benoit Baudry, David Benavides. A Variability-Based Testing Approach for Synthesizing Video Sequences. International Symposium on Software Testing and Analysis (ISSTA), Bay Area, California. **Core A conference.**

Chapter 7 *SPLC'14:* Mathieu Acher, Mauricio Alférez, **José A Galindo**, Pierre Romenteau, Benoit Baudry. ViViD: A Variability-Based Tool for Synthesizing Video Sequences (SPLC), Florence, Italy. **Main conference worldwide on software product lines.**

Chapter 8 *JSS'13:* Jules White, **José A. Galindo**, Tripti Saxena, Brian Dougherty, David Benavides, Douglas C. Schmidt. Evolving feature model configurations in software product lines. Journal of Systems and Software 11/2013.**Impact Factor: 1.245.**

Chapter 9 *IST'14:* **José A Galindo**, Deepak Dhungana, Goetz Botterweck, Rick Rabiser, Paul Grünbacher, David Benavides. Supporting Distributed Product Configuration by Integrating Heterogeneous Variability Modeling Approaches. Information and Software Technology (2nd round in revision).**Impact Factor: 1.328.**

These contributions while being related to the PhD candidate learning process are not intimately related within the thesis document.

- *JSS'14* Roberto E. Lopez-Herrejon, Lukas Linsbauer, **José A. Galindo**, José Á. Parejo, David Benavides, Sergio Segura, Alexander Egyed. An Assessment of Search-Based Techniques for Reverse Engineering Feature Models. Journal of Systems and Software (In press). **Impact Factor: 1.245**.
- *JLAMP'14* Ganesh Khandu Narwane, **José A. Galindo**, Shankara Narayanan Krishna, David Benavides, Jean-Vivien Millo, S Ramesh. Traceability analyses between features and assets in software product lines. Journal of Logical and Algebraic Methods in Programming (1st round in revision). **Impact Factor: 0.383**.
- *IST'14* Mauricio Alferez, **José A. Galindo**, Mathieu Acher, Benoit Baudry, David Benavides. Modeling Variability in the Video Domain: Language and Experience Report. Journal of Systems and Software (Prepared, this publication is related with chapter 7). **Impact Factor: 1.328**.

VU:

Le Directeur de Thèse

VU:

Le Responsable de l'École Doctorale

VU pour autorisation de soutenance
Rennes, le

Le Président de l'Université de Rennes 1

Guy CATHELINEAU

VU après soutenance pour autorisation de publication:

Le Président de Jury,

Résumé

Une particularité importante du logiciel est sa capacité à être adapté et configuré selon différents scénarios. Récemment, la variabilité du logiciel a été étudiée comme un concept de première classe dans différents domaines allant des lignes de produits logiciels aux systèmes ubiquitaires. La variabilité est la capacité d'un produit logiciel à varier en fonction de différentes circonstances. Les systèmes à forte variabilité mettent en jeu des produits logiciels où la gestion de la variabilité est une activité d'ingénierie prédominante. Les diverses parties de ces systèmes sont couramment modélisées en utilisant des formes différentes de "modèle de variabilité", qui est un formalisme de modélisation couramment utilisé. Les modèles de caractéristiques (feature models) ont été introduits par Kang et al. en 1990 et sont une représentation compacte d'un ensemble de configurations pour un système à forte variabilité.

Le grand nombre de configurations d'un modèle de caractéristiques ne permet pas une analyse manuelle. De fait, les mécanismes assistés par ordinateur sont apparus comme une solution pour extraire des informations utiles à partir de modèles de caractéristiques. Ce processus d'extraction d'information à partir de modèles de caractéristiques est appelé dans la littérature scientifique "analyse automatisée de modèles de caractéristiques" et a été l'un des principaux domaines de recherche ces dernières années. Plus de trente opérations d'analyse ont été proposées durant cette période.

Dans cette thèse, nous avons identifié différentes questions ouvertes dans le domaine de l'analyse automatisée et nous avons considéré plusieurs axes de recherche. Poussés par des scénarios du monde réel (e.g., la téléphonie mobile ou la vidéo protection), nous avons contribué à appliquer, adapter ou étendre des opérations d'analyse automatisée pour l'évolution, le test et la configuration de systèmes à forte variabilité.

Abstract

The large number of configurations that a feature model can encode makes the manual analysis of feature models an error prone and costly task. Then, computer-aided mechanisms appeared as a solution to extract useful information from feature models. This process of extracting information from feature models is known as *"Automated Analysis of Feature models"* that has been one of the main areas of research in the last years where more than thirty analysis operations have been proposed.

In this dissertation we looked for different tendencies in the automated analysis field and found several research opportunities. Driven by real-world scenarios such as smart phone or video–surveillance domains, we contributed applying, adapting or extending automated analysis operations in variability intensive systems evolution, testing and configuration.

This document was typeset on 2015/3/16 using RC-BOOK α2.11 for LaTeX$_{2\varepsilon}$. Should you want to use this document class, please send mail to contact@tdg-seville.info.