COFFEE: A FRAMEWORK SUPPORTING EXPRESSIVE VARIABILITY MODELING AND FLEXIBLE AUTOMATED ANALYSIS

Angela Villota

COFFEE: A FRAMEWORK SUPPORTING EXPRESSIVE VARIABILITY MODELING AND FLEXIBLE AUTOMATED ANALYSIS

ÁNGELA VILLOTA GÓMEZ

A Thesis Submitted for the Degree of Doctor of Philosophy in Computer Science

Centre de Recherche en Informatique
Université Paris 1 - Panthéon Sorbonne
Defended the 18th January, 2022

Members of the jury

Nicole LEVY	Laboratoire CEDRIC, CNAM	President
Camille SALINESI	Paris 1 Panthéon-Sorbonne University	Advisor
Raul MAZO	ENSTA Bretagne	Co-Advisor
David BENAVIDES	University of Sevilla	Reviewer
Roberto LOPEZ-HERREJÓN	University of Montreal	Reviewer
Don BATORY	The University of Texas at Austin	Examiner
Daniel DIAZ	Paris 1 Panthéon-Sorbonne University	Examiner

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Ángela Villota Gómez
18th January, 2022

In the loving memory of my abuelo Guillermo,
I was that curious, and stubborn tomboy because you were my accomplice.
You planted the seed, so I grew up to be the woman I become, I love you 3000.

"*A los que buscan*
aunque no encuentren.
A los que avanzan
aunque se pierdan.
A los que viven
aunque mueran."

Mario Benedetti

Abstract

Keywords: variability modeling, variability analysis, variability management, modeling languages, variability-intensive systems.

Modeling variability and automated analysis of variability models are tasks that come together. Nowadays, it is unrealistic to think about modeling variability without automated support for detecting model's anomalies and guide the configuration and derivation of products. The introduction of the FODA [KCH+90] language in the early 90's, marked the beginning of a fruitful development of languages, methods, and tools to describe the common and variable characteristics on families of systems. Also, the industrial application of variability modeling techniques on different types of variability-intensive systems came along with the automation of the analysis strategies. The size of the models, *i.e.,* the number of variable items and constraints, makes the configuration and the anomalies detection unfeasible tasks for humans. Consequently, there is a universe of languages, notations, transformations, algorithms, and tools supporting modeling and analysis tasks. Regretfully, this diversity brings issues and challenges in porting or sharing variability models among tools because of the lack of standards to represent variability. This diversity is even present among tools supporting variations of the same modeling language, as in feature-based notations, where the differences impact the expressiveness and the automation of analysis tasks.

At some point, during the exploration phase of the Ph.D., I wondered why the community that continually invests its efforts on providing methods, techniques, and tools for managing variability had somehow overlooked the reuse and variability management in their own proposals and contributions. It seems that the engineering of tools supporting variability modeling and analysis overlooks the reuse management and variability managing techniques. Tool engineers are continually re-implementing the transformations and analyses as they use specific representations, transformation rules, and technologies.

This thesis offers a variability management view to the tasks concerning variability modeling and analysis from the point of view of the Programming Languages Engineering. The Coffee Framework is a constraint-based framework that supports variability modeling and analysis about variability models, two of the main tasks in the management of variability-intensive systems. This thesis addresses the interoperability between variability management tools, the diversity among variability modeling languages, and the strong dependencies in the automated analysis of variability models. To solve these problems using intermediate representations, encodings, and transformations, this thesis presents two original contributions. First, this thesis defines, formalizes, implements, and evaluates an expressive variability modeling language (the High-Level Variability Language).

Second, this research includes a proposal of a three-step transformation framework to provide flexible, multi-language, and multi-solver support for automated analysis of variability models.

Traduction française du résumé

Mots-clés: modélisation de la variabilité, analyse de la variabilité, gestion de la variabilité, langages de modélisation, systèmes à forte variabilité.

La modélisation de la variabilité et l'analyse automatisée des modèles de variabilité sont des tâches qui vont de pair. De nos jours, il est irréaliste de penser à modéliser la variabilité sans un support automatisé pour détecter les anomalies du modèle et guider la configuration et la dérivation des produits. L'introduction du langage FODA [KCH+90] au début des années 90 a marqué le début d'un développement fructueux de langages, de méthodes et d'outils pour décrire les caractéristiques communes et variables des familles de systèmes. De même, l'application industrielle des techniques de modélisation de la variabilité sur différents types de systèmes à forte variabilité s'est accompagnée de l'automatisation des stratégies d'analyse. La taille des modèles, c'est-à-dire le nombre d'éléments variables et de contraintes, rend la configuration et la détection des anomalies des tâches irréalisables pour les humains. Par conséquent, il existe un univers de langages, de notations, de transformations, d'algorithmes et d'outils supportant les tâches de modélisation et d'analyse. Malheureusement, cette diversité entraîne des problèmes et des défis dans le portage ou le partage des modèles de variabilité entre les outils en raison du manque de normes pour représenter la variabilité. Cette diversité est même présente parmi les outils supportant des variations du même langage de modélisation, comme dans les notations basées sur les caractéristiques, où les différences ont un impact sur l'expressivité et l'automatisation des tâches d'analyse.

À un moment donné, pendant la phase d'exploration du doctorat, je me suis demandé pourquoi la communauté qui investit continuellement ses efforts pour fournir des méthodes, des techniques et des outils de gestion de la variabilité avait en quelque sorte négligé la réutilisation et la gestion de la variabilité dans ses propres propositions et contributions. Il semble que l'ingénierie des outils supportant la modélisation et l'analyse de la variabilité néglige la gestion de la réutilisation et les techniques de gestion de la variabilité. Les ingénieurs d'outils réimplémentent continuellement les transformations et les analyses lorsqu'ils utilisent des représentations, des règles de transformation et des technologies spécifiques.

Cette thèse propose une vision de la gestion de la variabilité pour les tâches concernant la modélisation et l'analyse de la variabilité. Coffee est un cadre basé sur les contraintes qui supporte la modélisation de la variabilité et l'analyse des modèles de variabilité, deux des principales tâches de la gestion des systèmes à forte variabilité. Cette thèse aborde l'interopérabilité entre les outils de gestion de la variabilité, la diversité entre les langages de modélisation de la variabilité et les fortes dépendances dans l'analyse automatisée des modèles de variabilité. Pour résoudre ces problèmes en utilisant des langages intermédiaires, cette thèse présente deux contributions originales. Premièrement, cette thèse définit, formalise, implémente et évalue un langage de modélisation de variabilité expressif (le *High-Level Variability Language*). Deuxièmement, cette recherche inclut une proposition d'un cadre de transformation en trois étapes pour fournir un support flexible, multi-langage et multi-solveur

pour l'analyse automatisée des modèles de variabilité.

Acknowledgments

Nine months before starting the PhD, I had the chance to stumble with a Philip H. Knight's quote that reads:

There comes a time in every life when the past recedes and the future opens. It's that moment when you turn to face the unknown. Some will turn back to what they already know. Some will walk straight ahead into uncertainty. I can't tell you which one is right. But I can tell you which one is more fun.

This quote well summarizes my journey, I've been facing the unknown in every sense, and I had a hell of fun!!

I want to thank my supervisor and co-supervisor Professors Camille Salinesi and Raúl Mazo. Camille thanks for your total support and confidence, for your caring and life advice, your insights, your always helpful remarks, and especially for teaching me the research path and its endeavors. Raúl thanks for your warm welcome to the lab, your teachings, your valuable insights about this project, and your friendship.

I want to thank the committee members, professors Nicole Levy, Daniel Diaz, David Benavides, Don Batory, and Roberto Erick Lopez-Herrejón for dedicating their invaluable time to evaluate my work. To the reviewers of this dissertation, David Benavides and Roberto Lopez-Herrejón, I appreciate your time in reviewing my dissertation and providing valuable feedback. Your helpful and insightful feedback will help me grow as a professional and researcher.

I would also like to thank the financial supporters of my PhD studies: Colfuturo, Universidad Icesi, and the Centre de RecheRche en Informatique (CRI) - University of Paris 1. Many thanks to my friends and colleagues at Universidad Icesi their encouragement, and continuous support were indispensable for the whole process.

I would like to thank to my colleagues and administrative staff at the CRI, professors Daniel Diaz, Benedicte Le Grande, Rebecca DeKenere, Jaques Robin, and all *des amies du labo!* (the lab's friends) Asmaa, Ali, David, Fabrice, Danny, Juan Carlos, Elena E., Elena K, Housseim, Sabrine, and Stephane. It was a pleasure to share with you.

To the friends I made in the research path, Don Batory, David Benavides, and Jose Galindo, thanks for your kind disposition, for always being there to answer my questions, and for all the talks and ideas.

To my family and friends, thank you for your support and confidence. Thanks to Lolo, Andresito, Fabio, and Juan Manuel for being there, for all the readings and talks, for listening and discussing

all my ideas and thoughts. Thanks to Jaime Chavarriaga for his generous guide and actively collaborating in my research. Special thanks to Luisa Rincón. I am aware of how fortunate I've been to find such a friend and partner in crime like her. I am very grateful for her presence in my life, and mere thanks are not enough. To Luisa, here's my promise to keep searching for new adventures together in the years to come.

To my beloved children, Juan Martín and Julieta, thank you for your support and patience, for all the times your beautiful smiles gave me comfort, strength, and courage. Thank you, my dears, for all the nights your cuddles gave me peace and rest.

And last but not least, thanks to my husband Gerardo M. Sarria M. for walking along with me, holding my hand. *Gracias Mau por ser la balandra de mis sueños, la gaveta donde guardo todos mis pensamientos, el cofre donde se esconde mi sonrisa, en donde moran mis ansias y mis recuerdos.*

Contents

List of Figures

List of Tables

Introduction en Français

Une fois que vous avez appris la variabilité, vous la voyez partout.

Ce chapitre fournit au lecteur un aperçu de la recherche présentée dans cette thèse. Pour commencer, le chapitre présente le contexte et délimite la portée de cette recherche et l'énoncé du problème. Les sections suivantes décrivent les détails de la recherche avec les questions de recherche, les objectifs de recherche et la méthodologie. Finalement, le chapitre se termine par un résumé des contributions et de la feuille de route de cette thèse.

Contexte

Les principes de la réutilisation des logiciels apparaissent dès les premières années du génie logiciel et sont en constante évolution. Par exemple, le concept de *familles de programmes* de David Parnas date de 1976 [Par76]. Dans ses travaux, Parnas envisageait déjà une philosophie de réutilisation lorsqu'il a décrit un processus de développement d'un ensemble de programmes en identifiant leurs caractéristiques communes et les conférant des propriétés individuelles. L'idée principale de la réutilisation est de prendre un élément tel qu'il est, sans le retraiter. Ainsi, les avantages de la réutilisation sont d'économiser du temps, de l'argent et des ressources. Ces avantages sont la raison pour laquelle la réutilisation est une alternative pour répondre à la demande croissante de systèmes logiciels complexes préservant la qualité et développés dans des délais de mise sur le marché plus courts.

Cependant, pour répondre à ces attentes, la réutilisation doivent être planifiées et axées sur la stratégie. Cette prise de conscience a propulsé l'essor des approches d'analyse de domaine pour l'identification systématique des caractéristiques communes dans les systèmes connexes. Le rapport de Kyo Kang, Feature Oriented Domain Analysis (FODA), constitue une étape importante dans l'analyse de domaine et la réutilisation systématique [KCH+90]. Dans ce rapport ont été étudiées les approches précédentes afin de définir une méthode et des outils pour aider les praticiens de l'analyse de domaine. Cependant, la contribution la plus importante du rapport de Kang est l'introduction des modèles de caractéristiques comme langage graphique pour décrire les caractéristiques et les fonctionnalités communes et variables dans une collection de systèmes logiciels apparentés.

Ce qui a commencé comme une bonne pratique, puis une méthode, est devenu un paradigme pour développer des systèmes logiciels. Le début des années 2000 a vu la consolidation des stratégies de réutilisation systématique dans l'Ingénierie des Lignes de Produits logiciels (SPLE par le sigle en anglais). Le SPLE est le paradigme permettant de produire des systèmes logiciels à grande échelle en utilisant une base technique commune et, en même temps, en répondant aux besoins individuels des clients.

Au cours des 20 dernières années, l'Ingénierie des Lignes de Produits Logiciel a suscité un intérêt considérable de la part de la communauté des chercheurs. Plusieurs publications font état de réalisations importantes et de l'expérience acquise lors de l'introduction de lignes de produits logiciels dans l'industrie du logiciel [MP14]. De nombreuses expériences réussies de l'application des principes de l'Ingénierie des Lignes de Produits Logiciels dans des contextes industriels sont répertoriées dans le *"Product Line Hall of Fame"* [1]. Des exemples de cas industriels réussis sont, par exemple, le cas de Boeing [Sha98, DS00], Hewlett-Packard [TCO00], et Lucent Technologies [ADD+00], entre autres. Selon Clements et Northrop [CN01], ces entreprises ont fait état d'avantages importants, par exemple, elles ont constaté des gains allant jusqu'à décupler la productivité et la qualité, une réduction des coûts allant jusqu'à 60 %, une diminution des besoins en main-d'œuvre allant jusqu'à 87 % et une réduction du délai de mise sur le marché (nouvelles variantes) allant jusqu'à 98 %. Plus les systèmes sont intensifs en logiciels, plus ils peuvent bénéficier de la stratégie SPLE.

La l'Ingénierie des Lignes de Produits Logiciel consiste à concevoir simultanément un ensemble de produits ou de services logiciels appelés lignes de produits. Une ligne de produits est une collection de produits similaires partageant des caractéristiques communes et répondant aux exigences d'une mission ou d'un segment de marché particulier. Les produits d'une ligne de produits logiciels sont assemblés à partir d'un ensemble commun d'actifs de base d'une manière prescrite [CN01]. Le concept de *variabilité* est au cœur de La SPLE car l'identification des points communs et de la variabilité est un prérequis majeur pour l'ingénierie des lignes de produits logiciels.

Definition 1 Variabilité *est la capacité d'un système, d'un actif ou d'un environnement de développement à supporter la production d'un ensemble d'artefacts qui diffèrent les uns des autres de manière planifiée [BC05].*

Cette définition est fréquemment associée à la *variabilité logicielle*. La variabilité logicielle permet de spécifier des artefacts logiciels qui ne sont pas entièrement définis au moment de la conception. Alors, la variabilité logicielle peut être interprétée comme un changement planifié ou anticipé [GWT+14]. La variabilité logicielle est légèrement différente de la *variabilité de la ligne de produit*.

Definition 2 Variabilité de la ligne de produits *décrit les variations entre les systèmes d'une ligne de produits. Ces variations peuvent se manifester en termes de fonctionnalités, de propriétés et d'exigences de qualité [MPH+07].*

La variabilité du logiciel et la variabilité de la ligne de produit sont gérées différemment. La variabilité logicielle peut être documentée sur les artefacts logiciels, les modèles et est prise en charge par la plupart des langages de modélisation et de programmation. Par exemple, pensez aux superclasses abstraites, aux interfaces facilitant différentes implémentations ou à la compilation conditionnelle (par exemple, à l'aide de #ifdefs) [MP14]. D'autre part, la définition de ce *qui varie* et de ce *comment il varie* au niveau de la ligne de produit nécessite des décisions explicites de la part de la direction du produit et des autres parties prenantes. L'ensemble des activités et des tâches nécessaires pour soutenir la spécification et la réalisation de la variabilité de la ligne de produits est appelé *gestion de la variabilité*.

Definition 3 Gestion de la variabilité *est l'ensemble des activités et des tâches permettant de définir et d'exploiter la variabilité tout au long du cycle de vie d'une ligne de produits logiciels [PBvdL05].*

[1] Disponible sur https://splc.net/fame.html

La gestion de la variabilité couvre les processus et les outils de modélisation, d'exploitation, de mise en œuvre et d'évolution de la variabilité. La modélisation de la variabilité permet et soutient la communication, la discussion, la gestion et l'analyse de la variabilité des lignes de produits. La modélisation de la variabilité consiste en la définition d'un modèle de variabilité. Les modèles de variabilité représentent explicitement les caractéristiques et fonctionnalités communes et variables des produits d'une ligne de produits. Ces modèles sont créés à l'aide de langages de variabilité. La recherche sur la modélisation de la variabilité comprend la proposition de plusieurs langages et outils de modélisation qui ont été proposés dans le milieu universitaire et l'industrie [BSL+13]. Le *modèles de caractéristiques* de Kang et ses notations dérivées sont le langage de variabilité le plus fréquemment utilisé. Depuis l'introduction du rapport FODA, plus de 40 dialectes différents de modèles de caractéristiques ont été proposés [BSRC10].

La recherche présentée dans cette thèse étudie la diversité dans la modélisation de la variabilité d'un point de vue ontologique et pratique. Du point de vue ontologique, cette thèse a étudié différents langages de modélisation pour définir un glossaire des concepts présents dans le langage de variabilité et étudie l'application de la théorie de l'expressivité ontologique aux langages de modélisation de la variabilité. D'un point de vue pratique, cette thèse présente un langage de modélisation de la variabilité textuelle visant *l'expressivité* du point de vue ontologique et *la flexibilité* pour représenter les concepts présents dans les langages de variabilité avec différents styles de modélisation. Cette thèse fournit une implémentation du langage en utilisant Xtext [Xte] et l'évaluation du langage en considérant la théorie de l'expressivité ontologique. L'évaluation a également pris en compte les scénarios et les besoins définis pour un langage de modélisation de la variabilité standard [BC19].

L'exploitation et la mise en œuvre de la variabilité sont des tâches qui nécessitent la transformation et le traitement des modèles de variabilité pour obtenir des informations. Cette transformation et ce traitement des modèles de variabilité sont connus sous le nom d'analyse de variabilité, une tâche importante pour la gestion de la variabilité. L'analyse de la variabilité englobe les méthodes, les outils et les techniques concernant la configuration, la vérification, les tests et la dérivation des modèles de variabilité [BSRC10]. Les techniques d'analyse ont fait l'objet d'études au cours des 20 dernières années. Un catalogue complet des techniques d'analyse peut être trouvé dans des revues de littérature telles que les travaux de Benavides *et al.* [BSRC10] et Galindo *et al.* [GBT+18].

La définition et l'implémentation du langage de modélisation dans cette thèse inclut également la formalisation de la sémantique opérationnelle et la conception et l'implémentation du cadre supportant l'analyse de variabilité. Il existe une relation intrinsèque entre la modélisation et l'analyse de variabilité, et un compromis bien documenté entre l'expressivité et le support d'analyse dans les outils de gestion de la variabilité. Ainsi, cette thèse inclut une proposition d'un cadre de transformation en trois étapes pour fournir une *flexibilité* concernant les règles de transformation, et les outils de résolution. Par conséquent, le cadre développé dans cette thèse est un cadre multi-langage et multi-solveur.

L'évolution des modèles de variabilité tient compte du fait que le développement et le déploiement de lignes de produits logiciels ne sont pas le résultat d'un effort ponctuel, mais le résultat d'un processus d'ingénierie itératif. Entre les itérations, les modèles de variabilité évoluent en introduisant des changements ou en étendant un des produits, ou toute la ligne de produits. Ainsi, les approches de gestion de la variabilité incluent également le support des tâches d'évolution, comme dans les travaux de Pleuss *et al.* [PBD+12], et de Montalvillo *et al.*

[MD16]. L'évolution des modèles de variabilité sort du cadre de cette thèse.

Enoncé du problème et domaine d'application

Les méthodes, techniques et outils de gestion de la variabilité ont été appliqués à des systèmes logiciels qui ne sont pas identifiés comme des lignes de produits logiciels mais qui traitent de la variabilité. Ces systèmes sont connus sous le nom de *variability-intensive* systèmes parce que la gestion de la variabilité est une activité d'ingénierie essentielle lors du développement de ces systèmes [Ben19]. Par exemple, considérons les écosystèmes logiciels, tels que l'écosystème Android [CN01], et les systèmes IoT auto-adaptatifs tels que, les systèmes d'irrigation intelligents [ASS+19], entre autres.

Le support des tâches de gestion de la variabilité pour les systèmes à forte variabilité pose des nouveaux défis dans le domaine. Cette thèse est liée aux défis concernant l'ingénierie des outils de gestion de la variabilité, en particulier, les préoccupations liées à la modélisation et l'analyse. La modélisation de la variabilité et l'analyse des modèles de variabilité sont les deux faces d'une même pièce. La modélisation de la variabilité sans analyse est inconcevable car les outils de modélisation aident le modélisateur à définir des modèles de haute qualité, sans défaut et maintenables. Cette assistance est obtenue par l'automatisation des tâches d'analyse telles que la détection des défauts, la correction du modèle et la configuration du produit, entre autres. Les sous-sections suivantes présentent les défis de la modélisation et de l'analyse de la variabilité.

Modélisation de la variabilité

La modélisation de la variabilité est un sujet largement étudié. La recherche dans ce domaine comprend plusieurs langages de modélisation de la variabilité qui ont été proposés dans le milieu universitaire et dans l'industrie [BSL+13]. La plupart des recherches dans ce domaine se concentrent sur les langages de modélisation basés sur les caractéristiques depuis l'introduction du FODA [KCH+90]. Cependant, d'autres approches de modélisation existent, comme les langages basés sur les points de variation [PBvdL05], les langages basés sur les décisions [DGR11], les langages orientés vers les objectifs [MFTR+15], et les langages basés sur les contraintes [SMDD10]. Il existe également des approches industrielles qui peuvent être utilisées pour décrire la variabilité, telles que Kconfig [ZC], Pure::variants [psG], Gears [Kru07], et le langage de modélisation *version-option* de Renault [ACF10]. Ces nombreuses propositions ont contribué à la création d'un univers de langages, de notations, de transformations et d'outils permettant la création de modèles de variabilité.

L'absence de standards est la principale raison pour laquelle la modélisation de la variabilité repose sur plusieurs langages et outils de modélisation spécifiques au domaine. La plupart de ces outils sont développés et enseignés *in-house*, et ne sont souvent utilisés que par les quelques personnes associées à l'équipe de développement.Cette diversité de langages et d'outils entraîne une faible portabilité des modèles et des problèmes d'interopérabilité entre les outils d'ingénierie SPLE. L'une des mauvaises conséquences est que les outils de modélisation nécessitent de nombreux analyseurs et transformations qui peuvent entraîner une perte d'expressivité.

Conscients de la nécessité d'un langage de variabilité standard qui capture de manière exhaustive la variabilité dans les systèmes à forte variabilité, au moins trois initiatives ont vu le jour ces dernières années. La première est la proposition du langage commun de variabilité (CVL) comme langage standard pour la modélisation de la

variabilité [Hau]. Cette initiative menée par Øystein Haugen, Andrzej Wasowski et Krzysztof Czarnecki est allée jusqu'à être considérée comme une norme par l'Object Management Group (OMG). Cependant, la tentative de définir le CVL comme un langage standard n'a pas été couronnée de succès en raison de conflits de droits d'auteur.

Une deuxième initiative est la spécification du Variability Exchange Language (VEL) [ApsGtFIfOCSF] en tant que langage générique pour échanger des informations sur la variabilité entre les outils de gestion de la variabilité et les outils de développement de systèmes. La spécification du VEL est l'un des résultats du projet SPES_XT qui réunit des partenaires industriels tels que Daimler AG, pure-systems GmbH et le Fraunhofer Institute for Open Communication Systems FOKUS. Une version préliminaire de la spécification de VEL est destinée à être normalisée au sein de l'Organization for the Advancement of Structured Information Standards (OASIS)[2].

La troisième initiative est plus récente. Au cours des deux dernières années, la communauté MODEVAR a été créée autour de l'idée de définir un langage textuel standard pour représenter les modèles de caractéristiques. Cette initiative est toujours en cours et des experts ont organisé un workshop pour discuter des questions telles que les scénarios d'utilisation, les problèmes d'expressivité et le support d'analyse afin de parvenir à un consensus. Il s'agit d'une communauté très active qui, à ce jour, a apporté trois contributions importantes à la standardisation des langages de modélisation des caractéristiques (1) une collection de scénarios d'utilisation d'un langage standard de modélisation des caractéristiques [BC19]; (2) un ensemble de *niveaux de langage* pour caractériser les concepts de modélisation et l'expressivité des langages basés sur les caractéristiques [TSS19]; et (3) une syntaxe concrète/abstraite pour un langage standard basé sur les caractéristiques proposé dans la thèse de maîtrise de Dominic Engelhardt [Eng20] et présenté à la dernière conférence SPLC [SFE+21].

Une leçon a été tirée de ces initiatives: le développement d'un langage standard pour la variabilité est un effort intensif car la communication de la variabilité à l'aide de modèles nécessite de prendre en compte trois niveaux conceptuels: le *paradigme de modélisation*, le *langage de modélisation*, et le *syntaxe concrète* de l'outil de modélisation.

Le paradigme de modélisation est l'approche adoptée pour créer un modèle. Tout comme les paradigmes de programmation, les paradigmes de modélisation déterminent les principales caractéristiques des différents langages de modélisation. Les paradigmes de modélisation de la variabilité sont, par exemple, la modélisation basée sur les caractéristiques, la modélisation basée sur les décisions, la modélisation basée sur les objectifs et la modélisation basée sur les contraintes, entre autres. Le paradigme de modélisation définit la sémantique de l'*unité de variabilité*, la *structure* du modèle et le *processus de configuration*. Voyons ces différences entre deux paradigmes, la modélisation basée sur les caractéristiques et la modélisation basée sur les décisions. Dans les langages basés sur les caractéristiques, les caractéristiques sont l'unité de variabilité, elles représentent les artefacts qui peuvent être inclus ou non dans une configuration. Les modèles de caractéristiques sont des structures arborescentes avec une racine unique et des relations inter-arborescentes fréquemment représentées textuellement pour une meilleure lisibilité. En revanche, les langages basés sur les décisions ont les décisions et les actifs comme unités de variabilité. Les modèles de décision sont des modèles de type graphis. Le processus

[2] https://www.oasis-open.org/

de configuration consiste à choisir parmi des options disponibles dans les décisions, à ouvrir ou à fermer d'autres décisions au cours du processus.

Le *language* définit l'ensemble des concepts de modélisation, des éléments syntaxiques, de la sémantique et des règles de bonne forme en tant qu'ensemble d'outils pour définir les modèles de variabilité. Les langages sont associés à un paradigme de modélisation. Par exemple, le Textual Variability Language (TVL) [CBH11] est un langage de modélisation basé sur les caractéristiques. Certains langages peuvent être considérés comme des extensions d'autres, comme le langage de caractéristiques basé sur les attributs de FeatureIDE [TKB+14], qui étend le langage FODA proposé par Kang [KCH+90].

La syntaxe concrète de l'outil correspond aux interprétations et à la mise en œuvre que les ingénieurs des outils donnent aux langages de modélisation de la variabilité. Contrairement à ce que l'on pourrait croire, il arrive que la syntaxe concrète des outils de modélisation ne soit pas exactement la même. Prenons par exemple les outils SPLOT et VariaMos. Ces deux outils prennent en charge les modèles de base, c'est-à-dire les modèles définis dans le rapport du FODA. Les arbres de caractéristiques de SPLOT sont des structures imbriquées similaires aux arbres de répertoires d'un gestionnaire de fichiers, avec des contraintes textuelles entre les arbres écrites sous forme d'expressions en CNF. En revanche, VariaMos fournit un toile de dessin pour spécifier les arbres de caractéristiques. VariaMos représente les caractéristiques en utilisant des ellipses plutôt que des rectangles comme dans d'autres outils. De plus, VariaMos introduit une construction graphique appelée *bundle* pour représenter les parents-enfants lorsqu'il y a plus d'une caractéristique enfant.

Figure 1: Illustration d'un exemple de modèles de variabilité de partage des conflits

Les problèmes concernant la portabilité et le partage des modèles de variabilité sont une conséquence de la variabilité à chacun des trois niveaux conceptuels de modélisation. La Figure 1 illustre un exemple de ces problèmes. Tout d'abord, considérons l'ingénieur Ⓐ et l'ingénieur Ⓒ, tous deux créent un modèle de variabilité en utilisant le même paradigme mais deux outils différents. Ils rencontrent des incompatibilités au

niveau de la *syntaxe concrète*, puisque l'un dessine un arbre avec des boîtes comme nœuds, et l'autre crée une structure arborescente imbriquée remplissant un formulaire dans un navigateur. À ce niveau, le partage n'est pas bidirectionnel car il est possible de télécharger des modèles SPLOT dans FeatureIDE mais pas dans l'autre sens. Les problèmes de partage à ce niveau peuvent être résolus en créant des analyseurs syntaxiques permettant le téléchargement de modèles écrits à l'aide d'autres outils. Cependant, en l'absence d'un langage commun pour le partage des modèles, les ingénieurs des outils doivent développer autant d'analyseurs syntaxiques qu'il existe d'outils.

Imaginons maintenant que le téléchargement de modèles dans les deux sens soit résolu. Il reste des problèmes pour permettre le partage de modèles entre l'ingénieur Ⓐ et l'ingénieur Ⓒ. Il y a toujours la possibilité que leurs modèles ne soient pas compatibles au niveau du *language*. Dans l'exemple, l'ingénieur Ⓐ utilise un outil supportant les attributs et les contraintes complexes dans ses modèles. Comme l'outil utilisé par l'ingénieur Ⓒ ne supporte pas les attributs ni les contraintes complexes, il est fort probable que l'analyse syntaxique des modèles de Ⓐ vers les modèles de Ⓒ entraîne une perte d'expressivité.

Dans l'exemple, les ingénieurs Ⓐ et Ⓒ utilisent des outils sous le même paradigme puisque tous deux créent des modèles de fonctionnalités. Considérons maintenant le problème lorsque le partage implique des modèles conçus sous différents paradigmes. Mettons dans l'image les ingénieurs Ⓑ et Ⓓ. L'ingénieur Ⓑ utilise REFAS [MFTR+15], un langage de modélisation dérivé des modèles de buts, conçu pour décrire la variabilité dans les systèmes auto-adaptatifs. En contraste, l'ingénieur Ⓓ utilise Dopler [DGR11], un langage de modélisation basé sur la décision. Au niveau du paradigme, le problème va au-delà de l'analyse syntaxique d'une syntaxe concrète à une autre, ou de la perte d'expressivité lors du passage d'un langage plus expressif à un langage moins expressif. Pour permettre le partage de modèles entre les ingénieurs Ⓑ et Ⓓ, il doit exister un encodage permettant de mapper les unités de variabilité dans les deux paradigmes, c'est-à-dire de transformer les *goals* en *decisions*, et de décrire les relations et les contraintes entre ces unités de variabilité. Ainsi, au niveau du paradigme, la complexité du problème augmente en raison du besoin de nombreux analyseurs syntaxiques et la perte d'expressivité dans le processus d'analyse syntaxique est toujours présente, mais les combinaisons dans le *mapping* entre les langages et outils compatibles explosent.

Pour résumer, la définition d'un langage de variabilité standard doit tenir compte de ce qui suit:

- La définition d'un élément variable abstrait capable de représenter des éléments variables de différents paradigmes de modélisation.
- La définition d'une structure flexible pour supporter des modèles avec des structures différentes.
- La définition d'un ensemble expressif de relations de variabilité et d'un langage de contraintes pour éviter la perte d'expressivité pendant l'analyse syntaxique.

Par conséquent, la conception d'un langage standard de modélisation de la variabilité doit tenir compte de la variabilité des langages actuels de modélisation de la variabilité en ce qui concerne les trois niveaux conceptuels, à savoir le paradigme, le langage et la syntaxe concrète. La dernière considération, mais non la moins importante, est celle des *personnes* car les standards ne peuvent pas être imposés. Les langages deviennent des standards parce que les gens les utilisent et parce que les ingénieurs d'outils incluent des fonctionnalités compatibles avec

les standards dans leurs outils. Il est particulièrement difficile de convaincre une communauté de passer d'une approche à une autre, surtout lorsqu'elle dispose de cas de succès avec des lignes de produits déployés résolvant les problèmes pour lesquels ils ont été conçus. L'adoption d'un nouveau langage de modélisation ne se fera pas tant que la nouvelle proposition n'aura pas prouvé son efficacité à résoudre le problème de l'échange de modèles et que les outils disponibles n'incluront pas l'exportation/importation de modèles dans une représentation standardisée.

Analyse de la variabilité

L'analyse de variabilité englobe les méthodes, les outils et les techniques utilisés pour extraire des informations des modèles de variabilité [BSRC10]. Les tâches d'analyse sont essentielles pour développer et maintenir les systèmes à forte variabilité, en particulier pour préserver la qualité du modèle qui a un impact important sur la qualité des produits d'une famille de produits. La communauté ELPS reconnaît que l'analyse de la variabilité est une tâche irréalisable pour les humains et qu'elle a donc besoin du soutien fourni par des mécanismes et des techniques automatisés. Dans les établissements industriels, les modèles de variabilité peuvent avoir des centaines ou des milliers d'éléments. Lorsque le nombre d'éléments variables dans un modèle augmente, le nombre de produits augmente de façon exponentielle. Par exemple, les gammes de véhicules produites par le constructeur français Renault peuvent donner lieu à 10^{21} de possibles configurations [ACF10]. Sans analyse automatisée, les outils de gestion de la variabilité ne sont que des applications de dessin spécialisées.

Au cours des 20 dernières années, plusieurs travaux ont contribué à doter les outils de gestion de la variabilité de mécanismes d'analyse pour automatiser différentes tâches. Par exemple, il existe des mécanismes d'analyse pour introduire des étapes dans le processus de configuration [WDSB09]; tester et gérer des lignes de produits [PSK+10]; détecter des anomalies [TBD+08, SM12]; trouver les causes de ces anomalies [FBGR13]; suggérer un ensemble de modifications pour supprimer les défauts [TBD+08]; et effectuer des opérations multimodèles [ACLF13, GDR+15] telles que la composition de modèles par des mécanismes de melange des modeles et d'agrégation. Benavides *et al.* présente un catalogue complet d'opérations d'analyse dans son étude systématique [BSRC10] poursuivie dans l'ouvrage de Galindo *et al.* [GBT+18].

Le support d'outils pour les techniques de modélisation et d'analyse de la variabilité profite de l'utilisation d'un cadre de transformation en deux étapes fréquemment signalé dans la littérature [MBC09, TKB+14, BSRC10, GBT+18]. La Figure 2 illustre le cadre de transformation en deux étapes adapté de [GBT+18].

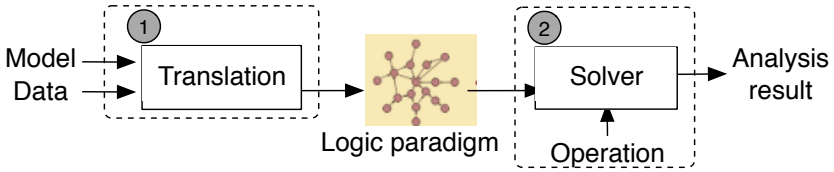


Figure 2: Cadre de transformation en deux étapes soutenant l'analyse de la variabilité. Adapté de [GBT+18].

Dans le premier pas décrit dans la Figure 1.2, les modèles de variabilité sont encodés dans une représentation logique en utilisant une collection de règles de transformation spécifiques. La représentation logique consiste en une collection de variables et une collection de relations entre ces variables représentées sous forme de formules

logiques ou de contraintes. Dans le second pas, un solveur *off-the-shelf* est utilisé pour résoudre le problème de satisfaction composé par la représentation logique et quelques contraintes supplémentaires pour produire un résultat d'une tâche d'analyse particulière. Il n'existe pas de représentation logique unique pour les modèles de variabilité. Les modèles de variabilité peuvent être représentés à l'aide de nombreux paradigmes logiques différents tels que les problèmes de satisfiabilité (SAT); [Bat05, MBC09, TKB+14]; Problèmes de satisfiabilité modulo théories (SMT) [UKB10, KAT16]; problèmes de satisfaction de contraintes (CSP) [BTRC05a]; et programmation logique avec contraintes (CLP) [SMD+11a, KOD13].

Si les modèles de variabilité sont représentés à l'aide de différents paradigmes logiques, ce n'est pas seulement parce que les chercheurs dans ce domaine sont continuellement à la recherche de représentations logiques mieux adaptées aux modèles de variabilité. Cette diversité de paradigmes logiques est principalement une conséquence du trade-off entre l'expressivité du langage de modélisation et la complexité du paradigme logique nécessaire pour représenter ces modèles [MP14, BSRC10, EKS13, GBT+18]. En d'autres termes, les langages plus expressifs posent des défis importants aux mécanismes d'analyse car ils nécessitent des paradigmes logiques plus complexes. Par exemple, les formules logiques en CNF codent les modèles de caractéristiques basiques de manière directe [MBC09]. En contraste, d'autres langages de modélisation supportant les attributs, la cardinalité des caractéristiques et les contraintes complexes requièrent des variables numériques et des contraintes arithmétiques et sont fréquemment encodés en tant que problèmes de satisfaction de contraintes [EKS13].

Deux situations problématiques naissent de cette diversité dans les représentations logiques. Premièrement, l'utilisation d'un paradigme logique plus complexe pour mieux capturer le langage de modélisation peut avoir un impact sur la performance du mécanisme d'analyse. Une représentation plus complexe d'un modèle de variabilité (encodage) contient généralement un plus grand nombre de variables avec des domaines plus larges et des contraintes plus complexes, c'est-à-dire des variables avec des domaines numériques et des expressions arithmétiques et relationnelles. Deuxièmement, différentes représentations logiques nécessitent différentes règles de transformation et donc différentes implémentations des opérations d'analyse. Ensuite, l'ingénierie des outils s'accompagne généralement d'une mise en œuvre interne de l'ensemble de la chaîne de transformation. Par conséquent, il existe plusieurs versions du même cadre d'analyse implémenté de différentes manières. La réimplémentation du cadre d'analyse contribue en quelque sorte au partage et à la portabilité des outils de gestion de la variabilité et néglige le consensus de la communauté concernant les opérations d'analyse, c'est-à-dire qu'il existe un catalogue [BSRC10] et une formalisation [DBS+17] de la sémantique des opérations d'analyse.

Après le choix d'un paradigme logique, l'étape suivante n'est pas simple. La forte dépendance entre l'expressivité du langage et le paradigme logique a un impact sur la sélection des technologies pour les outils de gestion de la variabilité de l'ingénierie. De plus, cette décision doit prendre en compte un large espace de possibilités entre les règles de transformation et les outils de résolution. Par exemple, les outils codant les modèles en tant que formules logiques peuvent utiliser des règles issues d'un large ensemble de propositions telles que Mannion [Man02], van Deursen [VK02], Batory [Bat05], et Mendonć a [MBC09], pour n'en citer que quelques-unes. Parallèlement, ces outils peuvent exploiter les caractéristiques des solveurs SAT ou BDD tels que SAT4J, Buddy, PicatSat et SPASS-SATT, entre autres. Ensuite, les questions *quel ensemble de règles de transformation* et *quels outils de résolution* doivent être incluse dans l'implémentation de la chaîne de transformation? sont des questions auxquelles on répond en fonction de l'expérience de l'équipe d'ingénieurs. Par conséquent, il existe plusieurs versions du cadre en deux étapes mises en œuvre de différentes manières, car

l'ingénierie des outils produit une mise en œuvre interne de l'ensemble de la chaîne de transformation.

Une leçon que j'appris en faisant partie de l'équipe d'ingénierie de la suite d'outils VariaMos [MSD12a, MMFR+15] au *Centre de Recherche en Informatique*, est que *la chaîne de transformation peut être réimplémentée plusieurs fois dans* un *outil* lorsqu'il intègre différents langages de variabilité ou différents solveurs même s'il n'y a qu'*un seul paradigme logique*. Les modèles de variabilité dans VariaMos sont représentés en utilisant la programmation logique par contraintes et peuvent être résolus avec n'importe quel solveur dérivé de Prolog. Cependant, l'ajout d'un nouveau langage de modélisation dans VariaMos implique la définition des règles d'encodage d'un modèle en tant que problème de contrainte en CLP. En plus, l'inclusion d'un nouveau solveur dérivé de Prolog dans l'outil demande la définition et l'implémentation de l'analyse d'un modèle dans sa représentation CLP en un code lisible par le solveur. Ce dernier point est dû au fait qu'il existe des différences dans la syntaxe concrète des solveurs dérivés de Prolog.

La prise de conscience des besoins et des avantages de la définition d'une norme pour les langages de modélisation de la variabilité a laissé de côté la diversité concernant le paradigme logique et les outils de résolution supportant les tâches d'analyse. Cependant, des efforts supplémentaires devraient être investis pour construire des repères concernant les règles de transformation et les outils de résolution ainsi que la construction collaborative d'une bibliothèque standard pour les tâches d'analyse qui profite des connaissances produites par la communauté de recherche. De plus, il est important de prendre en compte la diversité de ces préoccupations afin de pouvoir adapter la chaîne de transformation à de nouvelles représentations logiques, règles de transformation et outils de résolution.

Objectif et Questions de Recherche

Cette thèse aborde les problèmes concernant l'interopérabilité entre les outils de gestion de la variabilité, la diversité des langages de modélisation de la variabilité et le couplage fort dans le cadre supportant les tâches d'analyse. Pour contribuer à la résolution de ces problèmes du point de vue des langages intermédiaires, cette thèse aborde l'objectif de recherche suivant.

Objectif de recherche.

Explorer l'utilisation des langages intermédiaires pour faciliter l'interopérabilité des outils de gestion de la variabilité en concevant un cadre basé sur les contraintes supportant un langage de variabilité *expressif* et un mécanisme d'analyse automatisé *flexible*. Pour atteindre l'objectif de recherche, les trois questions de recherche suivantes ont été abordées.

Questions de recherche

RQ1: *Comment l'utilisation d'un langage unifié de modélisation de la variabilité peut faciliter l'interopérabilité entre les outils de gestion de la variabilité?*

La recherche présentée dans cette thèse développe l'hypothèse de l'utilisation de représentations intermédiaires comme mécanismes de découplage. Ainsi, la solution aux problèmes d'interopérabilité du point de vue de la

modélisation de la variabilité, considère la conception et l'implémentation d'un langage expressif de modélisation de la variabilité capable d'unifier les langages actuels et de représenter les relations de variabilité de manière exhaustive. Ce langage peut être utilisé comme une représentation intermédiaire pour encoder les langages de modélisation de la variabilité afin de partager et de porter les modèles entre les outils. L'expressivité est la caractéristique la plus importante du langage proposé car il doit englober les constructions de variabilité de différents langages et outils. En d'autres termes, les constructions du langage doivent être suffisantes pour encoder les modèles produits dans les outils actuels, en évitant toute perte d'expressivité dans le processus d'encodage. En outre, le langage doit fournir des constructions pour modéliser les relations de variabilité qui sont moins courantes mais potentiellement utiles pour décrire la variabilité. La RQ1 peut être affinée dans les sous-questions suivantes.

RQ1.1 D'un point de vue théorique, quelles sont les caractéristiques d'un langage expressif de modélisation de la variabilité?

RQ1.2 Quelles sont les caractéristiques d'un langage de modélisation de la variabilité pour être considéré comme un format d'échange dans différents outils de modélisation?

RQ1.3 Comment définir un langage de modélisation de la variabilité en évitant la dépendance de l'implémentation?

These subquestions will guide the design of an expressive variability modeling language centered on expressiveness, considering the community it is designed for, and the approach required to define a language whose syntax and semantics are independent from any particular implementation.

RQ2: *Comment les représentations intermédiaires supportent l'analyse automatisée des modèles de variabilité avec une approche flexible?*

Les cadres supportant l'analyse automatisée des modèles de variabilité sont limités par le compromis entre l'expressivité et les capacités d'analyse, et le couplage entre les paradigmes logiques et les outils de résolution. Cependant, les tâches d'analyse des modèles de variabilité sont essentielles pour la gestion de la variabilité. Donc, tout langage de modélisation nécessite la définition d'un cadre d'analyse. Le langage de modélisation que cette thèse se propose de définir n'est pas l'exception. En suivant l'hypothèse centrale d'exploiter les représentations intermédiaires des modèles de variabilité, cette thèse vise à développer un mécanisme d'analyse englobant l'expressivité du langage de modélisation d'une manière indépendante du solveur. Pour assurer l'indépendance du solveur, cette thèse exploite les avantages des différentes représentations logiques et des solveurs. La QR2 peut être raffinée dans les sous-questions suivantes.

RQ2.1 Quelle représentation logique est la mieux adaptée pour encoder des modèles de variabilité décrits avec un langage de modélisation expressif?

RQ2.2 Quel ensemble de règles de transformation devrait être considéré pour encoder les modèles de variabilité?

RQ2.3 Quelles sont les caractéristiques du langage abstrait utilisé pour encoder les modèles de variabilité en tant que problèmes de satisfaction de contraintes pouvant être résolus par différents solveurs?

Les réponses à ces questions sont utiles pour concevoir et évaluer l'approche de prise en charge des tâches d'analyse des modèles de variabilité dans le langage de modélisation de la variabilité hautement expressif.

RQ3: *Comment les représentations intermédiaires peuvent-elles être intégrées dans un cadre pour faciliter l'interopérabilité des outils de gestion de la variabilité?*

Les première et deuxième questions portent sur la manière de résoudre des problèmes particuliers concernant la modélisation et l'analyse de la variabilité. Cette question porte sur l'intégration des propositions de modélisation et d'analyse en tant que cadre pour la gestion de la variabilité. La RQ3 peut être affinée dans les sous-questions suivantes.

RQ3.1 Comment soutenir le flux de travail de la gestion de la variabilité, de la modélisation à la réponse à des questions d'analyse particulières?

RQ3.2 Comment les outils de gestion de la variabilité peuvent-ils interagir ou être intégrés dans le flux de travail proposé par le cadre?

Cette recherche étudie les modèles de modélisation et d'analyse de la variabilité comme deux préoccupations différentes. La RQ3 et ses sous-questions associées consolident ces préoccupations dans un cadre où la variabilité entre les outils de modélisation et d'analyse n'est pas un problème pour le partage et le portage des modèles de variabilité.

Méthode de recherche

Le travail de recherche global pour cette thèse a été exécuté selon les principes de la *Design Science Research.* (DSR). Les sous-sections suivantes présentent brièvement la DSR, décrivent les étapes de la recherche rapportée dans cette thèse et comment ces étapes répondent au processus et aux principes de la DSR.

Design Science Research

La DSR est un paradigme de recherche fréquemment utilisé pour la recherche dans le domaine des systèmes d'information. La DSR consiste en la conception et l'évaluation d'artefacts destinés à résoudre des problèmes organisationnels identifiés [HMPR04]. Ces artefacts sont développés et utilisés dans un contexte qui comprend des personnes, des organisations et des systèmes techniques (environnement du problème). Le cadre de recherche proposé par Hevner *et al.* dans [HMPR04] englobe trois cycles d'activités étroitement liés. La Figure 3 présente le cadre de recherche de la science du design tel que décrit par Hevner *et al.* dans [Hev07].

Le *cycle de conception* est au cœur du projet DSR. Ce cycle est une itération entre la construction et l'évaluation des artefacts de conception. La Figure 3 illustre le cycle de conception central et la manière dont il relie les cycles de pertinence et de rigueur. Le *cycle de pertinence* consiste à identifier et à représenter les opportunités et les problèmes dans l'environnement de l'application. Ce cycle produit l'ensemble des exigences utilisées comme critères d'acceptation pour évaluer les artefacts produits dans le processus de recherche. Le *cycle de fondement* fournit aux chercheurs les théories scientifiques, les méthodes d'ingénierie, les expériences

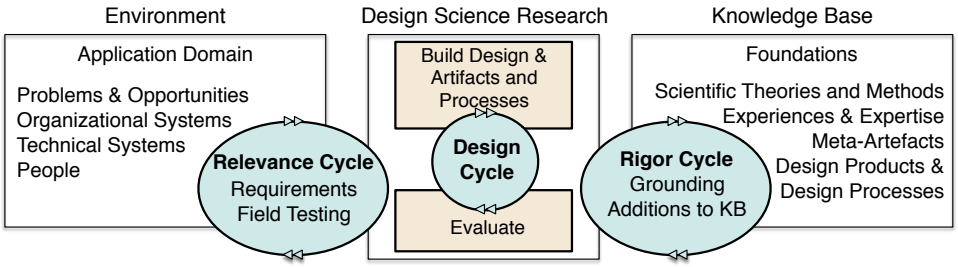


Figure 3: Design Science Research Cycles from [Hev07]

et l'expertise qui servent de fondements à la recherche. Ce cycle reçoit également les ajouts à la base de connaissances produits dans le cycle de conception.

Phases de recherche

Le projet de recherche présenté dans cette thèse comporte deux phases: la *phase d'exploration* et la *phase de définition*. Chaque phase constitue une itération du cycle de conception central exécuté pour répondre aux questions de recherche.

La phase d'exploration consistait en la conceptualisation, l'exploration et l'évaluation du cadre HLCL.

Le cadre HLCL tire son nom du High-Level Constraint Language, un langage abstrait permettant de représenter des modèles de variabilité à partir de différentes notations, développé au *Centre de Recherche en Informatique*. Le cadre HLCL est intégré aux premières implémentations de la suite d'outils VariaMos [MMFR+15] et a évolué grâce aux recherches de Djebbi & Salinesi, Salinesi *et al.*, Mazo *et al.* et Munoz-Fernandez *et al.*. [DS08, SMD+11a, MSD11, MFTR+15].

La motivation de commencer par une exploration du cadre HLCL est double. Premièrement, l'hypothèse de l'applicabilité des représentations intermédiaires développée dans cette thèse est compatible avec la prémisse d'inclure une étape supplémentaire pour représenter les modèles de variabilité en utilisant un langage de contraintes abstrait. Deuxièmement, le cadre HLCL s'est avéré utile pour représenter différents langages de variabilité [SMD+11a, MSD+12b, MFTR+15], et pour supporter la vérification et la configuration des modèles de variabilité dans ces langages [MSD+12b, SM12, MFTR+15]. Cependant, ce cadre et son implémentation dans la suite d'outils VariaMos font partie de l'écosystème des outils de gestion de la variabilité non compatibles. Ensuite, l'évaluation de l'étendue du cadre HLCL d'un point de vue pratique et formel était primordiale pour une meilleure compréhension du problème et de ses perspectives de solution. La phase d'exploration comprenait trois activités principales détaillées comme suit.

Activity 1. Conceptualisation et revue de la littérature. Cette recherche a débuté par une étude sur les cadres de transformation basés sur les contraintes prenant en charge les tâches de gestion de la variabilité. Cette conceptualisation visait à répondre aux questions sur les limites des cadres basés sur les contraintes pour la gestion de la variabilité. Voici la liste des activités incluses dans la conceptualisation.

A.1. 1. *Étude systématique de literature sur l'application des contraintes dans l'ingénierie des lignes de produits.* Le premier pas dans la conceptualisation a consisté à définir un cadre de classification pour réaliser une revue de literature systématique. Cette étude de literature a suivi les directives de l'Evidence-Based Software Engineering, en particulier les directives de Petersen *et al.* [PVK15]. L'objectif de cette étude est de comprendre la portée des approches basées sur les contraintes dans le cycle de vie des systèmes à forte variabilité. Le Chapitre 2 présente la conception, l'exécution et les résultats de cette étude.

A.1. 2. *Raffinement du cadre HLCL.* La deuxième étape de la conceptualisation a consisté à appliquer les concepts appris dans la définition d'un modèle conceptuel pour le cadre HLCL. Les modèles conceptuels sont des vues simplifiées abstraites d'un système utilisées pour aider les gens à comprendre ou à expliquer le système qu'ils représentent. Le cadre HLCL a été intégré dans l'outil VariaMos et présenté dans des publications précédentes. Cependant, ce cadre manquait d'une définition conceptuelle. Le raffinement de ce cadre est présenté dans le Chapitre 3.

Activity 2. Évaluation pratique. La deuxième étape de la phase d'exploration comprend la conception, l'implémentation et l'évaluation d'une notation graphique basée sur les contraintes pour décrire les modèles de variabilité. Cette notation graphique a été appelée *Graphes de Contraintes*. L'objectif de l'évaluation pratique consistait à expérimenter les difficultés rencontrées par un ingénieur lors de l'implémentation/extension des capacités de modélisation et d'analyse dans un outil de gestion de la variabilité particulier. A ce stade, j'ai conçu et développé l'inclusion des graphes de contraintes comme l'un des langages de modélisation dans la suite d'outils VariaMos et l'implémentation et l'évaluation d'un algorithme de vérification. Voici la liste des sous-activités incluses dans l'évaluation pratique.

A.2. 1. *Définition et mise en œuvre des graphes de contraintes comme notation graphique pour le HLCL.* Les graphes de contraintes représentent graphiquement les modèles de variabilité décrits en HLCL [MVSD]. Cette représentation graphique a été implémentée en JAVA et incluse dans la version *stand-alone* de la suite d'outils VariaMos en suivant l'approche de métamodélisation proposée par Munoz-Fernandez *et al.* [MFTR+15].

A.2. 2. *Implémentation et évaluation de MEDIC*[3]. MEDIC est un algorithme de vérification qui permet de détecter les contraintes incohérentes et de représenter graphiquement la manière dont ces contraintes sont liées. Ces informations guident les concepteurs dans leur prise de décision lors du débogage et de la correction des modèles de variabilité. MEDIC a été implémenté en JAVA et inclus dans la version stand-alone de la suite d'outils VariaMos. Une évaluation de la précision et des performances de MEDIC-PLM a été réalisée et nous avons fourni des preuves expérimentales que la méthode fournit rapidement les sources d'incohérences et qu'elle est extensible aux modèles industriels.

Les résultats de l'évaluation pratique ont été exclus de cette thèse car ils n'ont pas eu d'impact sur les décisions concernant la conception du cadre ou de l'une de ses parties. Par ailleurs, la proposition de graphe de contraintes, son implémentation, l'algorithme de vérification et son évaluation sont le sujet principal d'une publication à venir intitulée "MEDIC: Method to Diagnose Inconsistent Product Line Models using Constraint Graphs" qui sera soumise au journal of Requirements Engineering [MVSD].

[3]MEDIC est l'abréviation de Method to Diagnose Inconsistent Product Line Models using Constraint Graphs

Activity 3. Évaluation théorique. Cette étape est appelée évaluation théorique car elle est fondée sur la théorie de l'expressivité ontologique introduite par Wand & Weber [WW93]. La théorie de l'expressivité ontologique est un cadre d'évaluation bien fondé, également appliqué pour évaluer d'autres langages de modélisation tels que le modèle entité-relation [SMN+10]; UML [BJM08], le langage $i^\star$ [GFG12]; et BPMN [RRIG09].

A.3. 1. *Évaluation ontologique de l'expressivité de la HLCL en tant que langage de modélisation.* L'étape finale de la conceptualisation a consisté en la conception et l'exécution d'une évaluation ontologique visant à mesurer l'expressivité de la HLCL en tant que langage de modélisation. Cette évaluation a également pris en compte les métriques de Recker *et al.* pour évaluer l'expressivité ontologique par l'achèvement et la clarté: (1) le degré de déficit, (2) le degré d'excès, (3) le degré de redondance et (4) le degré de superposition [RRIG09]. Le Chapitre 3 présente la conception, l'exécution et les résultats.

Les résultats des évaluations ont montré plusieurs failles dans la proposition originale. L'approche de représentation intermédiaire dans le cadre HLCL est utile pour intégrer différents langages de variabilité dans un seul outil. Par exemple, VariaMos fournit une interface graphique pour décrire la variabilité d'un système unique en utilisant différentes vues et plus d'une notation [MMFR+15]. Toutefois, les avantages du cadre HLCL pour le support de différentes notations et de différents solveurs ne compensent pas les inconvénients concernant les facteurs clés suivants.

1. Le HLCL présente des imperfections, un manque de usabilite et de readabilite. À un certain point, l'utilisation de ce langage ressemble au remplacement d'un langage de programmation par un langage d'assemblage: quels que soient ses avantages, travailler avec des programmes d'assemblage à grande échelle sans un langage de plus haut niveau et plus abstrait est une tâche irréalisable. De plus, du point de vue de l'implémentation, l'approche consistant à fournir une représentation abstraite interne limite les contraintes, les fonctions et les opérateurs disponibles pour représenter les modèles de variabilité, limitant ainsi le niveau d'expressivité du langage de modélisation.

2. L'extensibilité de l'outil implémentant ce cadre, à savoir l'inclusion de nouveaux langages et de nouveaux solveurs, nécessite la définition et l'implémentation de nouvelles règles de transformation. L'ajout d'un nouveau langage de modélisation dans l'outil implique la définition des règles d'encodage des langages de variabilité en représentations HLCL. De plus, l'inclusion d'un nouveau solveur demande la définition des règles pour obtenir un encodage lisible par le solveur. Enfin, l'utilisation d'une représentation intermédiaire ne résout pas la réimplémentation de la chaîne de transformation.

3. La représentation intermédiaire des modèles de variabilité à l'aide de la HLCL n'offre pas une indépendance complète du solveur. En effet, l'utilisation de plusieurs solveurs dans la HLCL n'est pas pratique car elle nécessite une transformation supplémentaire pour encoder les représentations HLCL en code lisible par les solveurs. Ainsi, l'inclusion de nouveaux solveurs implique la définition et la mise en œuvre de nouvelles règles de transformation.

4. La représentation intermédiaire des modèles de variabilité à l'aide de HLCL ne résout pas les problèmes de portabilité et de partage des modèles entre les outils. Les modèles de variabilité initialement décrits dans différentes notations et ensuite encodés dans des représentations HLCL sont utiles pour être analysés

de manière intégrée dans le même outil [DTS+14, MFTR+15]. Toutefois, le cadre HLCL reproduit les problèmes de portabilité à l'intérieur de la layer de représentation intermédiaire comme une conséquence de la définition et de l'implémentation in-house de le langage. La HLCL est un sous-groupe de la programmation par contraintes contenant les opérateurs et les expressions employés pour encoder les modèles de variabilité. Par conséquent, une représentation standard de la programmation par contraintes pourrait être un meilleur candidat pour encoder les modèles en tant que contraintes d'une manière indépendante du solveur.

La deuxième itération du cycle de conception a pris en compte les résultats obtenus dans la phase d'exploration. Les paragraphes suivants décrivent comment les problèmes du cadre HLCL ont été abordés dans la conception, la mise en œuvre et l'évaluation du cadre présenté dans cette thèse.

La phase de conception est la deuxième itération du cycle de conception. Cette phase a couvert la conception, l'implémentation et l'évaluation du cadre Coffee. Ce cadre traite séparément les préoccupations de modélisation et les préoccupations d'analyse comme deux parties d'un seul cadre. La conception du cadre exploite l'hypothèse développée dans cette thèse selon laquelle l'introduction de représentations intermédiaires facilite l'interopérabilité entre les outils et le couplage dans les implémentations des outils. Ensuite, la deuxième itération englobe les activités suivantes. Notez que les activités énumérées ci-dessous n'ont pas été réalisées de manière séquentielle.

Activity 4. Modélisation conceptuelle du cadre Coffee Le modèle conceptuel du cadre a été proposé et affiné de manière itérative au cours de la *phase de conception*. Le Chapitre 3 décrit le modèle conceptuel et présente ses composants.

Activity 5. Conception et évaluation du High-Level Variability Language (HLVL) HLVL est la propose de cette thèse pour un langage textuel de modélisation de la variabilité suffisamment expressif et flexible pour être considéré comme un langage intermédiaire pour échanger des modèles et faciliter la portabilité des modèles et les problèmes de partage entre les outils de modélisation. La conception et l'évaluation ont consisté en les sous-activités suivantes.

A.5. 1. *Definition de la syntaxe, de la sémantique et des caractéristiques de HLVL.* Les décisions prises dans la conception de HLVL sont les conséquences des résultats et des enseignements tirés de la phase d'exploration. La conception du langage a pris en compte les concepts de modélisation recueillis lors de la revue de la littérature, les critères d'expressivité ontologique, la caractérisation des langages de variabilité textuelle, les différents paradigmes de modélisation et les échanges avec les experts du domaine. Le Chapitre 4 détaille HLVL et ses caractéristiques.

A.5. 2. *Évaluation ontologique de l'expressivité de la HLVL.* La dernière étape de la définition du langage de modélisation a consisté à effectuer une évaluation ontologique. Cette évaluation a pour objectif de garantir l'expressivité du HLVL (*cf.* Chapitre 5). Ainsi, le langage proposé serait capable d'encoder la plupart des langages de modélisation de la variabilité sans perte d'expressivité.

Activity 6. Conception et évaluation du cadre d'analyse. Après avoir abordé le problème de la représentation des modèles de variabilité à l'aide d'un langage intermédiaire sans perte d'expressivité, l'étape suivante consistait à fournir un cadre d'analyse des modèles HLVL. Les sous-activités réalisées pour définir, mettre en

œuvre et évaluer le cadre d'analyse sont les suivantes.

A.6. 1. *Définition de la sémantique opérationnelle de HLVL*. L'idée générale qui sous-tend la sémantique opérationnelle est de définir les étapes du calcul de la sortie à partir de segments de programme ou de code [Plo81, Läm18]. Donc, la sémantique opérationnelle de HLVL est une collection de règles de transformation appliquées pour transformer un modèle en une représentation logique pour effectuer des questions de satisfiabilité sur les solveurs. Le Chapitre 4 présente la sémantique opérationnelle pour chaque construction du langage en considérant différentes représentations logiques et différents solveurs.

A.6. 2. *Démonstration de la faisabilité du cadre d'analyse sensible au contexte.* La définition du cadre d'analyse a été complétée par la définition de l'architecture et l'implémentation d'un prototype. La précision de la solution a été mesurée en comparant l'ensemble des configurations obtenues avec une implémentation de Coffee et celles obtenues avec d'autres outils de gestion de la variabilitè. Le Chapitre 5 décrit les particularités de l'implémentation et discute des résultats obtenus.

Résumé des contributions

Cette thèse aborde les problèmes d'interopérabilité entre les outils de gestion de la variabilité, la diversité des langages de modélisation de la variabilité et les fortes dépendances dans l'analyse automatisée des modèles de variabilité. Pour résoudre ces problèmes en utilisant des langages intermédiaires, cette thèse présente les contributions originales suivantes:

1. Une étude systématique de literature sur le sujet des outils de gestion de la variabilité supportés par des cadres basés sur les contraintes (Chapitre 2). Pour cela, un cadre d'analyse et de comparaison de la littérature associée est développé, composé de multiples facettes concernant les concepts de modélisation de la variabilité traduits en contraintes, les systèmes de contraintes utilisés pour encoder les modèles de variabilité, les règles de transformation pour encoder les modèles de variabilité en tant que problèmes de satisfaction de contraintes et les solveurs supportant les tâches d'analyse.

2. Un cadre théorique pour évaluer l'expressivité des langages de variabilité du point de vue ontologique. Ce cadre d'évaluation est fondé sur la théorie de l'expressivité ontologique [WW93] et son application aux langages de modélisation de la variabilité proposée par Asadi *et al.*. [AGWH12]. Le cadre d'évaluation s'appuie également sur l'ontologie fondamentale de la variabilité incluse dans les travaux de Reinhartz-Berger *et al.*. [RBSW11] et les métriques permettant de déterminer les critères de complétude et de clarté introduites par Recker *et al.* [RRIG09]. Ce cadre a été appliqué pour évaluer l'expressivité de la programmation par contraintes pour représenter les modèles de variabilité (Chapitre 3) et le langage de modélisation proposé dans cette thèse.

3. La définition d'un cadre de transformation basé sur les contraintes appelé Coffee (Chapitre 3). La définition et l'implémentation de ce cadre intègrent les contributions précédentes, en traitant les problèmes de variabilité et d'analyse séparément, mais en considérant ces problèmes comme faisant partie d'une seule proposition. Coffee est un cadre à quatre niveaux contenant une collection de méthodes et d'artefacts permettant d'encoder les modèles de variabilité en HLVL, d'encoder les modèles HLVL en problèmes de

satisfaction de contraintes, et d'effectuer des tâches d'analyse en utilisant l'infrastructure indépendante du solveur.

4. La conception et la définition formelle d'un langage textuel de modélisation de la variabilité appelé langage de variabilité de haut niveau, HLVL en abrégé (Chapitre 4). L'objectif principal de la conception du HLVL est de fournir aux ingénieurs de domaine un langage textuel riche pour modéliser la variabilité dans des contextes académiques et industriels. Les principales caractéristiques du langage sont sa *expressivité* du point de vue ontologique et sa *flexibilité* pour représenter les concepts présents dans les langages de variabilité avec différents styles de modélisation. Ce langage peut être utilisé comme langage de modélisation de la variabilité et comme représentation intermédiaire d'autres langages de variabilité. L'idée générale est que les modèles de variabilité peuvent être encodés en HLVL comme une forme intermédiaire et peuvent être interprétés sur d'autres outils réduisant les problèmes d'interopérabilité et de partage.

5. La définition de la sémantique opérationnelle de HLVL sous la forme de règles d'inférence considérant le contexte de la transformation (Chapitre 4). Cette sémantique opérationnelle traite du compromis entre expressivité et analysabilité en encodant les modèles HLVL en problèmes de satisfaction à l'aide de deux paradigmes logiques. De plus, les modèles de variabilité sont encodés dans une seconde représentation intermédiaire utilisant une représentation générique des contraintes qui fournit une solution indépendante du solveur. Ainsi, cette proposition prend en compte différents niveaux d'expressivité, règles de transformation, paradigmes de résolution et outils de résolution.

Publications

Cette thèse étend et réutilise le contenu d'articles publiés, en cours de révision ou en préparation. Chaque chapitre de cette thèse fait référence à la publication ou au manuscrit correspondant. La liste des publications est la suivante:

- Ángela Villota, Raúl Mazo, and Camille Salinesi. A CP-Based Framework for Product Line Engineering. Forum Jeunes Chercheurs du congrés INFORSID (2016). Actes du 8 e Forum Jeunes Chercheurs congrés INFORSID.
- Raul Mazo, Ángela Villota, Camille Salinesi and Daniel Diaz. MEDIC: Method to Diagnose Inconsistent Product Line Models using Constraint Graphs. To be submitted to the Requirements Engineering journal.
- Ángela Villota, Raúl Mazo, and Camille Salinesi. On the Ontological Expressiveness of the High-Level Constraint Language for Product Line Specification. In: Khendek F., Gotzhein R. (eds) System Analysis and Modeling. Languages, Methods, and Tools for Systems Engineering. SAM 2018. Lecture Notes in Computer Science, vol 11150. Springer, Cham. DOI: https://doi.org/10.1007/978-3-030-01042-3_4
- Ángela Villota, Raúl Mazo, and Camille Salinesi. 2019. The High-Level Variability Language: An Ontological Approach. In Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B (SPLC '19). Association for Computing Machinery, New York, NY, USA, 162-169. DOI: https://doi.org/10.1145/3307630.3342401.

- Ángela Villota, Raúl Mazo,Camille Salinesi and Daniel Diaz. Constraints in Software Product Line Engineering: a Classification Framework and Systematic Mapping Study. Manuscript in preparation.

Outils

Trois outils différents ont été développés au cours du doctorat. Ces outils font partie des artefacts développés à différentes étapes de la recherche, comme indiqué dans la section . Le tableau 1 présente les liens vers les dépôts GitHub des outils.

Table 1: Résumé des outils

Outil	Disponible sur
MEDIC	Dossier personnel, https://github.com/angievig/VARIAMOS, https://github.com/angievig/MEDIC_TESTS Dossier du VariaMos, https://github.com/SPLA/VARIAMOS
HLVL Editor	Dossier personnel, https://github.com/angievig/Coffee/tree/master/HLVL
Coffee Platform	Dossier du projet, https://github.com/coffeeframework

MEDIC. La conception, la mise en œuvre, l'évaluation, le rapport et le déploiement de la méthode de diagnostic des modèles de lignes de produits incohérents à l'aide de graphiques de contraintes (MEDIC). Ce travail a été réalisé au cours des premières étapes du doctorat, lorsque je développais une représentation graphique pour le langage de contraintes de haut niveau. À la suite de cette première exploration, j'ai inclus les graphes de contraintes comme notation pour décrire la variabilité dans la suite d'outils VariaMos. De plus, j'ai développé MEDIC, un algorithme de diagnostic pour détecter les ensembles incohérents sur un modèle de variabilité spécifié comme un graphe de contraintes. L'algorithme de diagnostic a également été intégré dans la version autonome de la suite d'outils VariaMos. De plus, l'expérimentation et le benchmark liés à la conception et au développement de MEDIC font partie d'un article de journal à venir.

HLVL Editor. Cet outil prend en charge la grammaire et la sémantique opérationnelle d'une première version du langage de variabilité de haut niveau présenté et discuté lors de la conférence MODEVAR. Cet éditeur a été implémenté en java en utilisant la technologie Xtext.

Coffee Platform. Pour prendre en charge l'analyse automatisée associée à différents langages de variabilité, une architecture microservices de Coffee a été dessinée et développée. Dans cette version microservices du cadre, chaque niveau a été mis en œuvre à l'aide de Java et d'une API REST, puis placé dans un conteneur pour être déployé dans le swarm Docker situé à l'université Icesi de Cali. La conception et la mise en œuvre de la

plateforme de microservices Coffee font partie d'un projet financé par l'Université Icesi avec la collaboration d'étudiants du laboratoire i2t.

Feuille de route de la dissertation

Cette dissertation contient six chapitres divisés en trois parties, suivis d'une annexe contenant des informations auxiliaires.

Partie 1, Motivation et Contexte. Cette partie regroupe les chapitres délimitant le contexte de la recherche, la méthode, et l'état de la recherche associée à cette thèse.

CH 2. État de la recherche présente une étude systématique et un cadre de classification de la recherche existante liée aux cadres basés sur les contraintes supportant la modélisation et l'analyse de la variabilité. Ce chapitre présente un résumé des concepts de modélisation de la variabilité, des règles de transformation pour encoder les modèles dans des paradigmes logiques, et des solveurs utilisés dans l'analyse automatisée des modèles de variabilité.

Partie 2, Etudes et Résultats. La deuxième partie de cette thèse présente deux chapitres contenant les études concernant l'objectif principal de cette thèse: l'expressivité de la modélisation de la variabilité et le support d'analyse aux langages expressifs de modélisation de la variabilité.

CH 3. De l'évaluation du High-Level Constraint cadre au cadre Coffee résume les résultats obtenus lors de l'évaluation du cadre HLCL réalisée pendant la phase d'exploration. Ce chapitre se concentre sur la conception, l'exécution et les résultats de l'analyse ontologique des contraintes comme langage de modélisation de la variabilité. Et plus particulièrement, comment les résultats obtenus ainsi que l'expérience acquise lors de la phase d'exploration ont contribué à la modélisation conceptuelle du cadre Coffee. Le chapitre se termine par une présentation générale du cadre.

CH 4. Modélisation de la variabilité et analyse de la variabilité dans Coffee présente le langage de variabilité de haut niveau (High-Level Variability Language -HLVL), sa définition formelle, sa sémantique opérationnelle et le flux de travail allant des modèles de variabilité au processus d'analyse. Tout d'abord, le Chapitre 4 introduit le langage, décrit ses caractéristiques, présente la syntaxe et la sémantique formelles du langage. Deuxièmement, le chapitre explique comment Coffee prend en charge l'analyse automatisée en utilisant un cadre de transformation en trois étapes pour fournir un support flexible, multi-langage et multi-solveur pour l'analyse automatisée des modèles de variabilité spécifiés en HLVL. Le cadre de transformation proposé dans ce chapitre, est décrit sous la forme de règles d'inférence dans la définition de la sémantique opérationnelle de HLVL.

Partie 3, Analyse des résultats, discussion et perspectives. La dernière partie de cette thèse analyse et discute les résultats.

CH 5. Evaluation, Discussion, et Perspectives présente les évaluations utilisées pour démontrer l'expressivité du langage de modélisation et la flexibilité du cadre supportant l'analyse de variabilité.

CH 6. Conclusions clôt cette dissertation avec un résumé des résultats par rapport à chaque question de recherche, discute de l'impact de cette recherche, et présente les perspectives pour les travaux futurs.

Annexe. L'annexe A présente des informations supplémentaires produites lors de l'exécution de l'étude de literature systématique.

Résumé

Ce chapitre a présenté le contexte, l'énoncé du problème, les défis relevés, les objectifs et un aperçu des contributions de cette thèse.

Part I

Motivation and context

Chapter 1

Introduction

Once you learn about variability, you see variability everywhere.

This chapter provides the reader with an overview of the research presented in this thesis. To start, the chapter presents the context and delineates the scope of this research and the problem statement. The following sections describe the particulars of the research with the research questions, research objectives, and methodology. Finally, the chapter closes with a summary of the contributions and road map of this dissertation.

1.1 Context

The principles of software reuse appear in the early years of software engineering and are continually evolving. For instance, David Parnas' *program families* concept dates from 1976 [Par76]. In his work, Parnas already considered a reuse philosophy when he described a process for developing a set of programs identifying their common characteristics and providing them with individual properties. The main idea of reuse is to take an item as it is, without reprocessing. Thus, the advantages of reuse are to save time, money, and resources. These advantages are the reason why reuse is an alternative to answer the increasing demand of quality-preserving complex software systems developed in shorter times-to-market.

However, to meet these expectations, reuse strategies should be planned and strategy-driven. This realization propelled the rise of *domain analysis* approaches for the systematic identification of common characteristics in related systems. A milestone on domain analysis and systematic reuse is Kyo Kang's Feature Oriented Domain Analysis (FODA) report [KCH+90]. This report studies previous approaches to define a method and tools to support domain analysis practitioners. Yet, the most important contribution of Kang's report is the introduction of Feature Models as the graphical language to describe the common and variable characteristics and functionalities in a related collection of software systems.

What started as a good practice, then a method, became a paradigm to develop software systems. The early 2000's saw the consolidation of systematic reuse strategies into Software Product Line Engineering (SPLE). SPLE is the paradigm to produce software systems at a large-scale using a common technical base and, at the same time, answering individual customer's needs [PBvdL05].

In the past 20 years, SPLE has attracted significant interest from the research community. Several publications report significant achievements and experience gained by introducing software product lines in the software industry [MP14]. Many successful experiences of the application of SPLE principles on industrial settings are listed on "Product Line Hall of Fame" [1]. Examples of successful industrial cases are, for instance, the case of Boeing [Sha98, DS00], Hewlett-Packard [TCO00], and Lucent Technologies [ADD+00], among others. According to Clements and Northrop [CN01], these companies reported important benefits, for example, they found gains by as much as tenfold in productivity and quality, cost reduction by as much as 60%, decrease labor needs by as much as 87%, and decrease time to market (new variants) by as much as 98%. The more software-intensive systems become, the more they can benefit from the SPLE strategy.

SPLE consists in simultaneously engineering a collection of software products or services called product lines. A product line is a collection of similar products sharing common characteristics and satisfying the requirements of a particular mission or market segment. Products in a software product line are assembled from a common set of core assets in a prescribed way [CN01]. Central to SPLE is the concept of *variability* because the identification of commonality and variability is a major prerequisite for software product line engineering.

Definition 1.1 Variability *is the ability of a system, an asset, or a development environment to support the production of a set of artifacts that differ from each other in a preplanned fashion [BC05].*

This definition is frequently associated with *software variability*. Software variability allows the specification of software artifacts that are not fully defined at design time. Then, software variability can be interpreted as planned or anticipated change [GWT+14]. Software variability is slightly different from *product line variability*.

Definition 1.2 Product line Variability *describes the variations between systems in a product line. These variations can be in terms of functionalities, properties, and quality requirements [MPH+07].*

Software variability and product line variability are managed differently. Software variability can be documented on software artifacts, models and is supported by most modeling and programming languages. For instance, consider abstract superclasses, interfaces facilitating different implementations, or conditional compilation (e.g., using #ifdefs) [MP14]. On the other hand, defining *what does* vary and *how it varies* at the product line level requires explicit decisions from product management and other stakeholders. The set of activities and tasks needed to support the specification and realization of product line variability is called *variability management*.

Definition 1.3 Variability Management *is the set of activities and tasks supporting the definition and exploitation of variability throughout the life-cycle of a software product line [PBvdL05].*

Variability management covers the processes and tools for modeling, exploiting, implementing, and evolving variability. Modeling variability empowers and supports the communication, discussion, management, and analysis of product line variability [MP14]. Modeling variability consists on the definition of a variability model. Variability models explicitly represent the common and variable characteristics and functionalities among products in a product line. These models are created using variability languages. Research on variability modeling includes the proposal of several languages and modeling tools that have been proposed in academia

[1] Available at https://splc.net/fame.html

and industry [BSL+13]. Kang's *feature models* and its derived notations are the most frequently used variability language. Since the introduction of the FODA report, over 40 different feature model dialects have been proposed [BSRC10].

The research presented in this dissertation studies the diversity in variability modeling from an ontological and practical point of view. From the ontological point of view, this thesis studied different modeling languages to define a glossary of concepts present in variability language and studies the application of the theory of ontological expressiveness to variability modeling languages. From a practical point of view, this thesis presents a textual variability modeling language aiming *expressiveness* from the ontological point of view and *flexibility* to represent concepts present in variability languages with different modeling styles. This thesis provides one implementation of the language using Xtext [Xte] and the language's evaluation considering the theory of ontological expressiveness. The evaluation also considered the scenarios and needs defined to for a standard variability modeling language [BC19].

Exploiting and implementing variability are tasks that require the transformation an processing of variability models to obtain information. This transformation and processing of variability models is known as variability analysis, a significant task for managing variability. Variability analysis encircles the methods, tools, and techniques concerning the configuration, verification, testing, and derivation of variability models [BSRC10]. Analysis techniques have been the subject of study in the last 20 years. A comprehensive catalog of analysis techniques can be found in literature reviews such as the works of Benavides *et al.* [BSRC10] and Galindo *et al.* [GBT+18].

The definition and implementation of the modeling language in this thesis includes also the formalization of the operational semantics and the design and implementation of the framework supporting variability analysis. There exist an intrinsic relation between modeling and variability analysis, and a well documented trade-off between expressiveness and analysis support in variability management tools. Thus, this dissertation includes a proposal of a three-step transformation framework to provide *flexibility* regarding the transformation rules, and solving tools. Therefore, the framework developed in this thesis is a multi-language and multi-solver framework.

The evolution of variability models considers that to develop and deploy software product lines is hardly the output of a one-shot effort, but the result of an iterative engineering process. Between iterations, variability models evolve by introducing changes or extending one of the products, or all the product line. Thus, variability management approaches also include the support of evolution tasks, as in the works of Pleuss *et al.* [PBD+12], and Montalvillo *et al.* [MD16]. The evolution of variability models it out of the scope of this thesis.

The research presented in this dissertation is grounded on the theory, concepts and techniques from the Programming Languages Engineering. Then, the conception, design, and implementation of the framework included the development of intermediate representations, encodings, and transformations.

1.2 Problem statement and scope

Variability management methods, techniques, and tools have been applied to software systems that are not identified as software product lines but deal with variability. These systems, are known as *variability-intensive* systems because variability management is a core engineering activity when developing these systems [Ben19].

For example, consider software ecosystems, such as the Android ecosystem [CN01], and self-adaptive IoT systems such as, smart irrigation systems [ASS+19], among others.

The support of variability management tasks for variability-intensive systems poses new challenges in the area. This thesis is related to the challenges concerning the engineering of variability management tools, particularly, the concerns related to modeling and analysis. Modeling variability and analyzing variability models are two sides of the same coin. Variability modeling without analysis is inconceivable because modeling tools assist the modeler to define a high-quality, defect-free, and maintainable models. This assistance is achieved by the automation of analysis tasks such as the defect detection, model correction and product configuration, among others. The following subsections present the challenges on variability modeling and variability analysis.

1.2.1 Variability Modeling

Variability modeling is an extensively studied subject. Research in this subject includes several variability modeling languages that have been proposed in academia and industry [BSL+13]. Most research in the area focuses on feature-based modeling languages since the introduction of FODA [KCH+90]. However, other modeling approaches exist, such as variation point-based languages [PBvdL05], decision-based languages [DGR11], goal-oriented languages [MFTR+15], and constraint-based languages [SMDD10]. Also, there exists industrial approaches that can be used to describe variability, such as Kconfig [ZC], Pure::variants [psG], Gears [Kru07], and Renault's version-option modeling language [ACF10]. These numerous proposals have contributed to a universe of languages, notations, transformations, and tools supporting the creation of variability models.

The lack of standards is the main cause that variability modeling relies upon the several existing domain-specific languages and modeling tools. Most of these tools, are developed and taught in-house, and frequently are used only by the few people associated with the development team. This diversity of languages and tools causes low portability in models and interoperability issues between SPL engineering tools. One of the poor consequences is that modeling tools require numerous parsers and transformations that might cause expressiveness loss.

Well aware of the need for a standard variability language that comprehensively captures the variability in variability-intensive systems, at least three initiatives had emerged in the past years. The first is the proposal of the Common Variability Language (CVL) as a standard language for variability modeling [Hau]. This initiative led by Øystein Haugen, Andrzej Wasowski, and Krzysztof Czarnecki went as further as to be considered as a standard by the Object Management Group (OMG). However, the attempt to define CVL as a standard language did not succeed given to copyright conflicts.

A second initiative is the specification of the Variability Exchange Language (VEL) [ApsGtFIfOCSF] as a generic language to exchange variability information between variability management tools and systems development tools. The specification of VEL is one of the results of the SPES_XT that gathers industrial partners such as Daimler AG, pure-systems GmbH, and the Fraunhofer Institute for Open Communication Systems FOKUS. A draft version of the specification of VEL is intended to be standardized within the

Organization for the Advancement of Structured Information Standards (OASIS)[2].

The third initiative is more recent. In the past two years, the MODEVAR community[3] was created around the idea to set a standard textual language to represent feature models. This initiative is still a work in process where experts have organized a workshop to discuss issues such as usage scenarios, expressiveness issues, and analysis support to get a consensus. This is a very active community that to the day has three important contributions to the standardization of feature modeling languages (1) a collection of scenarios of usage of a standard feature modeling language [BC19]; (2) a set of *language levels* for characterize the modeling concepts and expressiveness of feature-based languages [TSS19]; and (3) a concrete/abstract for a standard feature-based language proposed in the master thesis of Dominic Engelhardt [Eng20] and presented in the latest SPLC conference [SFE+21].

There is one lesson learned from these initiatives: developing a standard language for variability is an effort-intensive endeavor because the communication of variability using models requires to consider three conceptual levels: the *modeling paradigm*, the *modeling language*, and the *concrete syntax* of the modeling tool.

The *modeling paradigm* is the approach to create a model. Similarly to programming paradigms, modeling paradigms define the main characteristics of the different modeling languages. Variability modeling paradigms are, for instance, feature-based modeling, decision-based modeling, goal-based modeling, and constraint-based modeling, among others. The modeling paradigm defines the semantics of the *variability unit*, the *structure* of the model, and the *configuration process*. Let's see these differences between two paradigms, feature-based modeling and decision-based modeling. In feature-based languages, features are the variability unit, they represent artifacts that can be included or not in a configuration. Feature models are tree-like structures with a single root with cross-tree relations frequently represented textually for better readability. In contrast, decision-based languages have decisions and assets as variability units. Decision models are graph-like models composed of decisions and assets linked to those decisions. The configuration process consists of deciding about the available options in the decisions, opening, or closing other decisions during the process.

The *language* defines the set of modeling concepts, syntactic elements, semantics, and well-formedness rules as toolset for defining variability models. Languages are associated with a modeling paradigm. For example, the Textual Variability Language (TVL) [CBH11] is a feature-based modeling language. Some languages can be considered extensions of others, as the attribute-based feature language of FeatureIDE [TKB+14], which extends the FODA language proposed by Kang [KCH+90].

The *concrete syntax* of the tool corresponds to the interpretations and implementation that tool-engineers give to variability modeling languages. Contrary to what could be expected, sometimes, the concrete syntax among modeling tools is not exactly the same. For example, consider the SPLOT and VariaMos tools. Both tools support basic-feature models, *i.e.,* as defined in the FODA report. Feature trees in SPLOT are nested structures in a similar way to directory trees in a file manager with textual cross-tree constraints written as expressions in CNF. In contrast, VariaMos provides a drawing canvas to specify feature trees. VariaMos depicts features using ellipses instead of rectangles as in other tools. Moreover, VariaMos introduces a graphical construct called

[2] https://www.oasis-open.org/

[3] https://modevar.github.io/

bundle to represent parent-children when there are more than one children features.

Figure 1.1: Illustration of an example of the conflicts sharing variability models

The issues regarding the portability and sharing of variability models are a consequence of the variability at each of the three modeling conceptual levels. Figure 1.1 illustrates an example of these issues. First, consider engineer Ⓐ and engineer Ⓒ, both of them are creating a variability model using the same paradigm but two different tools. They are experiencing incompatibilities at the *concrete syntax* level, since one is drawing a tree with boxes as nodes, and the other is creating a nested tree-like structure filling a form in a browser. At this level, the sharing is not bidirectional as it is possible to upload SPLOT models in FeatureIDE but not in the other direction. Sharing problems at this level can be solved by creating parsers enabling the upload of models written using other tools. However, without an common language for sharing models, tool engineers must develop as many parsers as tools exist.

Now, let's imagine that the upload of models in both directions is solved. There are still issues to allow the sharing of models between engineer Ⓐ and engineer Ⓒ. There is always the chance that their models are not compatible at the *language* level. In the example, engineer Ⓐ uses a tool supporting attributes and complex constraints within his models. Since the tool used by engineer Ⓒ does not support attributes nor complex constraints, there is a big chance the parsing from Ⓐ's models to Ⓒ's models causes expressiveness loss.

In the example, engineers Ⓐ and Ⓒ use tools under the same paradigm since both are creating feature models. Now, consider the problem when the sharing involves models designed under different paradigms. Let' include in the picture engineers Ⓑ and Ⓓ. Engineer Ⓑ uses REFAS [MFTR+15], a modeling language derived from goal models, designed to describe variability in self-adaptive systems. In contrast, engineer Ⓓ uses Dopler [DGR11], a decision-based modeling language. At the paradigm level, the problem goes beyond the parsing between a concrete syntax to another, or the expressiveness loss when moving from a more expressive language to a less expressive one. To allow the sharing of models between engineers Ⓑ and Ⓓ there must

exist an encoding for mapping the variability units in both paradigms, *i.e.,* transform *goals* into *decisions*, and to describe the relations and constraints between those variability units. Thus, at the paradigm level the complexity of the problem increases because of the need for numerous parsers and the expressiveness loss in the parsing process still holds but the combinations in the mapping among compatible languages and tools explodes.

To summarize, the definition of a standard variability language must consider:

- The definition of an abstract variable item capable of representing variable items from different modeling paradigms.
- The definition of a flexible structure to support models with different structures.
- The definition of an expressive set of variability relations and constraint language to avoid expressiveness loss during the parsing.

In consequence, the design of a standard variability modeling language must consider the variability of current variability modeling languages regarding the three conceptual levels, *i.e.,* paradigm, language and concrete syntax. A last but not least consideration is *people* because standards cannot be imposed. Languages become standards because people use them and tool engineers include standard-compatible features in their tools. It is particularly difficult to convince a community to move from one approach to another, specially when they have successful cases with product lines deployed solving the problems they were designed for. The adoption of a new modeling language wont happen until the new proposal has proven to be effective to solve the model exchange problem and when the available tools include the export/import of models in a standardized representation.

1.2.2 Variability Analysis

Variability analysis encircles the methods, tools, and techniques used to extract information from variability models [BSRC10]. Analysis tasks are central to develop and maintain variability intensive systems, particularly, to preserve the quality of the model which highly impacts the quality of products in a product family. The SPLE community acknowledges that variability analysis is an unfeasible task for humans and therefore requires the support provided by automated mechanisms and techniques. In industrial settlements, variability models may have hundreds or thousands of elements, as the number of variable items in a model grows, the number of products increases exponentially. For instance, the vehicle product lines produced by the French manufacturer Renault can lead to 10^{21} configurations [ACF10]. Without automated analysis, variability management tools are just specialized diagramming applications.

Over the past 20 years, several works have contributed to provide variability management tools with analysis mechanisms to automate different task. For instance, there exist analysis mechanisms for introducing steps in the configuration process [WDSB09]; testing and managing product lines [PSK+10]; detecting anomalies [TBD+08, SM12]; finding the causes of such anomalies [FBGR13]; suggesting a set of changes to remove defects [TBD+08]; and performing multimodel operations [ACLF13, GDR+15] such as composing models through merge and aggregation mechanisms. Benavides *et al.* present a comprehensive catalog of analysis operations in their systematic review [BSRC10] continued in Galindo *et al.*'s work [GBT+18].

Tooling support for variability modeling and analysis techniques profit the usage of a two-step transformation

framework frequently reported in the literature [MBC09, TKB+14, BSRC10, GBT+18]. Figure 1.2 illustrates the two-step transformation framework adapted from [GBT+18].

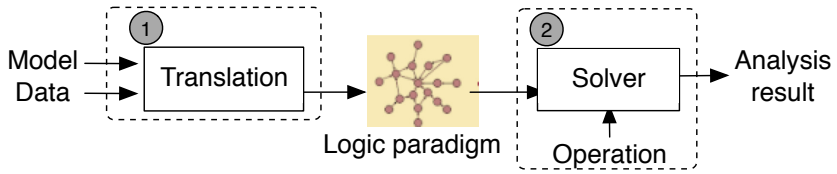


Figure 1.2: Two-step transformation framework supporting variability analysis. Adapted from [GBT+18].

In the first step depicted in Figure 1.2, variability models are encoded into a logic representation using a collection of specific transformation rules. The logic representation consists of a collection of variables and a collection of relations between these variables represented as logic formulas or constraints. In the second step, one *off-the-shelf* solver is used to solve the satisfaction problem composed by the logic representation and some extra constraints to produce a result of a particular analysis task. There is no a unique logic representation for variability models. Variability models can be represented using many different logic paradigms such as as Satisfiability Problems (SAT) [Bat05, MBC09, TKB+14]; Satisfiability Modulo Theories (SMT) problems [UKB10, KAT16]; Constraint Satisfaction Problems (CSP) [BTRC05a]; and Constraint Logic Programming (CLP) [SMD+11a, KOD13].

Variability models are represented using different logic paradigms not just because researchers in this area are continually searching for logic representations better suited for variability models. This diversity of logic paradigms, is mainly a consequence of the trade-off between the expressiveness of the modeling language and the complexity of the logic paradigm needed for represent such models [MP14, BSRC10, EKS13, GBT+18]. That is, more expressive languages pose significant challenges to analysis mechanisms because they require more complex logic paradigms. For instance, logic formulas in CNF encode basic feature models straightforwardly [MBC09]. In contrast, other modeling languages supporting attributes, feature cardinality, and complex constraints require integer variables and arithmetic constraints and are frequently encoded as constraint satisfaction problems [EKS13].

Two problematic situations derive from this diversity in logic representations. First, the usage of a more complex logic paradigm to better capture the modeling language may impacts the performance of the analysis mechanism. A more complex representation of a variability model (encoding) usually contains a greater number of variables with larger domains and more complex constraints, *i.e.,* variables with integer domains and arithmetic and relational expressions. Second, different logic representations require different transformation rules and therefore different implementations of the analysis operations. Then, tool engineering usually comes with an in-house implementation of the entire transformation chain. Therefore, there exist several versions of the same framework implemented in different ways. The reimplementation of the analysis framework somehow contributes to the sharing and portability of variability management tools and overlooks the community consensus regarding analysis operations, *e.g.,* there is a catalog [BSRC10] and a formalization [DBS+17] of the semantics of the analysis operations.

After a logic paradigm is chosen, the next step is not straightforward. The strong dependency between the language's expressiveness and the logic paradigm impacts the selection of technologies for engineering variability

management tools. Moreover, this decision must consider a broad space of possibilities between transformation rules and solving tools. For instance, tools encoding models as logic formulas can use rules from a large set of proposals such as Mannion [Man02], van Deursen [VK02], Batory [Bat05], and Mendonć a [MBC09] to mention some. At the same time, these tools can exploit the characteristics of SAT or BDD solvers such as SAT4J, Buddy, PicatSat, and SPASS-SATT, among other solvers. Then, the questions *which set of transformation rules* and *which solving tools* should be included in the implementation of the transformation chain? are questions that are answered regarding the experience of the engineering team. In consequence, there exist several versions of the two-step framework implemented in different ways as tool engineering produces an in-house implementation of the entire transformation chain.

A lesson I learned as part of the engineering team of the VariaMos tool-suite [MSD12a, MMFR+15] at the *Centre de Recherche en Informatique*, is that *the transformation chain can be re-implemented several times in* one *tool* when it integrates different variability languages or different solvers even if there is only *one logic paradigm*. Variability models in VariaMos are represented using constraint logic programming and can be solved with any of the Prolog-derived solvers. However, the addition of a new modeling language in VariaMos involves the definition of the rules for encoding a model as a constraint problem in CLP. In addition, the inclusion of a new Prolog-derived solver in the tool demands the definition and implementation for parsing a model in its CLP representation into a solver-readable code. The late because there exist differences in the concrete syntax of the Prolog-derived solvers.

The awareness about the needs and advantages of defining a standard for variability modeling languages left behind the diversity concerning the logic paradigm and solving tools supporting analysis tasks. However, further efforts should be invested to build benchmarks regarding transformation rules and solving tools as well as the collaborative construction of a standard library for analysis tasks that profits from the knowledge produced by the research community. Moreover, it is important to consider the diversity in these concerns to be able to adapt the transformation chain to new logic representations, transformation rules and solving tools.

1.2.3 Research Objective and Research Questions

This thesis addresses the problems regarding the interoperability between variability management tools, the diversity among variability modeling languages, and the strong coupling in the framework supporting analysis tasks. To contribute in solving these problems from an intermediate languages point of view, this thesis addresses the following research objective.

Research Objective.

Explore the usage of intermediate languages to ease the interoperability of variability management tools by designing a constraint-based framework supporting an *expressive* variability language and a *flexible* automated analysis mechanism. To achieve the research objective, the following three research questions were addressed.

Research Questions

RQ1: *How the usage of a unified variability modeling language can ease the interoperability among variability management tools?*

The research presented in this dissertation elaborates over the hypothesis of the usage of intermediate representations as decoupling mechanisms. Thus, the solution to interoperability problems from the variability modeling point of view, considers the design and implementation of an expressive variability modeling language able to unify current languages and represent variability relations in a comprehensive way. This language should be used as modeling language and as an intermediate representation to encode variability modeling languages for sharing and porting models among tools. Expressiveness is the most important characteristic of the proposed language because it must encompass the variability constructs from different languages and tools. That is, the language constructs must be sufficient to encode models produced in current tools avoiding expressiveness loss in the encoding process. Also, the language should provide constructs to model variability relations than are less common but potentially useful for describing variability. RQ1 can be refined in the following subquestions.

RQ1.1 From a theoretical perspective, what are the characteristics of an expressive variability modeling language?

RQ1.2 What are the characteristics of a variability modeling language for being considered as exchange format in different modeling tools?

RQ1.3 How to define a variability modeling language avoiding the implementation dependency?

These subquestions will guide the design of an expressive variability modeling language centered on expressiveness, considering the community it is designed for, and the approach required to define a language whose syntax and semantics are independent from any particular implementation.

RQ2: *How intermediate representations support the automated analysis of variability models with a flexible approach?*

The frameworks supporting automated analysis of variability models are limited by the trade-off between expressiveness and analysis capabilities, and the coupling among logic paradigms and the solving tools. However, analysis tasks over variability models are critical for variability management. Thus, any modeling language requires the definition of an analysis framework. The resulting modeling language this thesis intends to define is not the exception. Following the central hypothesis of exploiting intermediate representations of variability models, this thesis aims to develop an analysis mechanism encompassing the expressiveness of the modeling language in a solver-independent way. To provide solver independence, this thesis exploits the advantages of different logic representations and solvers. RQ2 can be refined in the following subquestions.

RQ2.1 Which logic representation is best suited for encoding variability models described with an expressive modeling language?

RQ2.2 Which set of transformation rules should be considered to encode variability models?

RQ2.3 What are the characteristics of the abstract language used for encoding variability models as constraint satisfaction problems solvable by different solvers?

The answers to these questions are useful to design and evaluate the approach for supporting analysis tasks to analyze variability models in the high-expressive variability modeling language.

RQ3: *How intermediate representations can be integrated in a framework to ease the interoperability of variability management tools?*

The first and second questions are about how to solve particular problems about variability modeling and analysis. This question is about the integration of the modeling and analysis proposals as a framework for variability management. RQ3 can be refined in the following subquestions.

RQ3.1 How to support the variability management workflow from modeling to answering particular analysis questions?

RQ3.2 How can variability management tools interact or be integrated into the proposed workflow of the framework?

This research studies the modeling and variability analysis models as two different concerns. RQ3 and its associated subquestions consolidate these concerns in one framework where the variability among modeling and analysis tools is not an issue for sharing and porting variability models.

1.3 Research method

The overall research work for this thesis has been executed under the principles of *Design Science Research* (DSR). The following subsections briefly presents DSR, describe the stages of the research reported in this thesis and how those stages meet the DSR process and principles.

1.3.1 Design Science Research

DSR is a research paradigm frequently used for research in the Information Systems domain. DSR consists in the design and evaluation of artifacts intended to solve identified organizational problems [HMPR04]. These artifacts are developed and used in a context that includes people, organizations, and technical systems (problem environment). The research framework proposed by Hevner *et al.* in [HMPR04] encompasses three closely related cycles of activities. Figure 1.3 presents the Design Science Research framework as depicted by Hevner *et al.* in [Hev07].

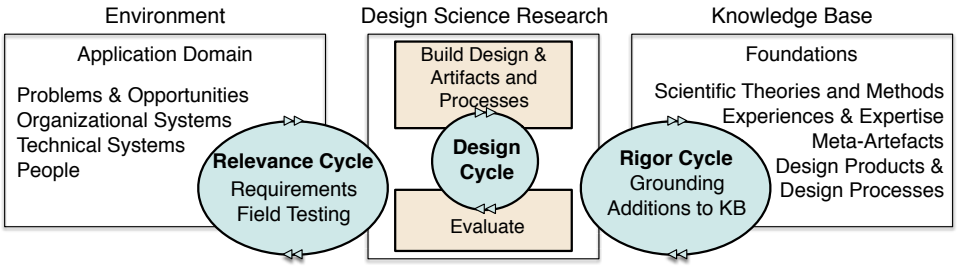


Figure 1.3: Design Science Research Cycles from [Hev07]

The *design cycle* is the core of the DSR project. This cycle iterates between the construction and evaluation of design artifacts. Figure 1.3 illustrates the central design cycle and how it connects both, the relevant and the rigor cycles. The *relevance cycle* consists in identifying and representing opportunities and problems in the

application environment. This cycle produces the set of requirements used as acceptance criteria to evaluate the artifacts produced in the research process. The *rigor cycle* provides researchers with the scientific theories, engineering methods, experiences, and expertise that serves as research foundations. This cycle also receives the additions to the knowledge base produced in the design cycle.

1.3.2 Research Phases

The research project reported in this thesis contains two phases: the *exploration phase* and the *definition phase*. Each phase constitutes one iteration of the central design cycle executed to answer the research questions.

The exploration phase consisted of the conceptualization, exploration, and evaluation of the state-of-the-art HLCL framework. The HLCL framework takes its name from the High-Level Constraint Language, an abstract language to represent variability models from different notations developed at the *Centre de Recherche en Informatique*. The HLCL framework is embedded on early implementations of the VariaMos tool-suite [MMFR+15] and evolved through the research of Djebbi & Salinesi, Salinesi *et al.*, Mazo *et al.*, and Munoz-Fernandez *et al.* [DS08, SMD+11a, MSD11, MFTR+15].

The motivation to start with an exploration of the HLCL framework is twofold. First, the hypothesis of the applicability of intermediate representations developed in this thesis is compatible with the premise to include one extra-step to represent variability models using an abstract constraint language. Second, the HLCL framework had proven useful for representing different variability languages [SMD+11a, MSD+12b, MFTR+15], and for supporting verification and configuration of variability models in those languages [MSD+12b, SM12, MFTR+15]. However, this framework and its implementation in the VariaMos tool suite are part of the ecosystem of non-compatible variability management tools. Then, the evaluation of the extent of the HLCL framework from a practical and formal perspective were primordial to a better understanding of the problem and its perspectives for solution. The exploration phase included three main activities detailed as follows.

Activity 1. Conceptualization and literature review. This research started with a study on constraint-based transformation frameworks supporting variability management tasks. This conceptualization aimed to answer questions about the limitations of constraint-based frameworks for variability management. The following is the list of activities included in the conceptualization.

A.1. 1. *Systematic mapping study on the application of constraints in product line engineering.* The first step in the conceptualization consisted of defining a classification framework to perform a systematic mapping study. This mapping study followed the guidelines from the Evidence-Based Software Engineering, particularly the guidelines from Petersen *et al.* [PVK15]. The aim of this study is to understand the reach of constraint-based approaches in the life-cycle of variability intensive systems. Chapter 2 presents the design, execution and results of this study.

A.1. 2. *Refinement of the HLCL framework.* The second step in the conceptualization was to apply the learned concepts in the definition of a conceptual model for the HLCL framework. Conceptual models are abstract simplified views of a system used to help people to understand or explain the system it represents. The HLCL framework had been embedded in the VariaMos tool and introduced in previous publications.

However, this framework lacked of a conceptual definition. The refinement of this framework is presented in Chapter 3.

Activity 2. Practical evaluation. The second stage in the exploration phase includes designing, implementing, and evaluating a constraint-based graphical notation for describing variability models. This graphical notation was called *Constraint Graphs*. The practical evaluation's objective consisted of experiencing the difficulties faced by a tool engineer when implementing/extending the modeling and analysis capabilities in a particular variability management tool. At this stage, I designed and developed the inclusion of constraint graphs as one of the modeling languages in the VariaMos tool-suite and the implementation and evaluation of a verification algorithm. The following is the list of sub-activities included in the practical evaluation.

A.2. 1. *Definition and implementation of constraint graphs as a graphical notation for HLCL.* Constraint graphs graphically represent variability models described in the HLCL [MVSD]. This graphical representation was implemented in JAVA and included in the stand alone version of the VariaMos tool-suite following the metamodeling approach proposed by Munoz-Fernandez *et al.* [MFTR+15].

A.2. 2. *Implementation and evaluation of MEDIC*[4]. MEDIC is a verification algorithm designed to detect inconsistent constraints and graphically depict how these constraints are related. This information guides designers on decision making when debugging and correcting variability models. MEDIC was implemented in JAVA and included in the stand alone version of the VariaMos tool-suite. An evaluation of the accuracy and performance of MEDIC-PLM was performed and we provided experimental evidence that the method quickly provides the sources of inconsistencies and it is scalable to industry models.

The results of the practical evaluation were excluded from this dissertation since they did not impacted the decisions about the design of the framework or any of its parts. Also, the constraints graph proposal, its implementation, the verification algorithm and its evaluation are the main subject of an incoming publication titled "MEDIC: Method to Diagnose Inconsistent Product Line Models using Constraint Graphs" to be submitted to the journal of Requirements Engineering [MVSD].

Activity 3. Theoretical evaluation. This step is called theoretical evaluation because it is grounded on the theory of ontological expressiveness introduced by Wand & Weber [WW93]. The theory of ontological expressiveness is a well-founded evaluation framework also applied to evaluate other modeling languages such as the entity-relation model [SMN+10]; UML [BJM08], the $i^\star$ language [GFG12]; and BPMN [RRIG09].

A.3. 1. *Ontological evaluation of the expressiveness of the HLCL as a modeling language.* The final step in the conceptualization consisted in the design and execution of an ontological evaluation aiming to measure the expressiveness of the HLCL as a modeling language. This evaluation also considered Recker *et al.*'s metrics to evaluate ontological expressiveness through completion and clarity: (1) degree of deficit, (2) degree of excess, (3) degree of redundancy, and (4) degree of overlap [RRIG09]. Chapter 3 presents the design, execution, and results.

The results of the evaluations showed several flaws in the original proposal. The intermediate representation approach in the HLCL framework is useful for integrating different variability languages in one single tool.

[4]MEDIC is short for Method to Diagnose Inconsistent Product Line Models using Constraint Graphs

For example, VariaMos provides a graphical interface to describe variability of a single system using different views and more than one notation [MMFR+15]. However, the benefits of the HLCL framework for supporting different notations and solvers do not compensate for the drawbacks regarding the following key factors.

1. The HLCL exhibits incompleteness, lack of usability, and readability. At some point, to use this language resembled replacing a programming language by assembly language: regardless of its benefits, to work with large scale assembly programs without a higher level, more abstract language is an unfeasible task. Additionally, from the implementation point of view, the approach to provide an in-house abstract representation limits the constraints, functions, and operators available to represent variability models, therefore, limiting the expressiveness level of the modeling language.

2. The extensibility of the tool implementing this framework, namely the inclusion of new languages and new solvers, requires the definition and implementation of new transformation rules. The addition of a new modeling language in the tool involves the definition of the rules for encoding variability languages into HLCL representations. Additionally, the inclusion of a new solver demands the definition of the rules for obtaining a solver-readable encoding. Then, the usage of an intermediate representation does not solve the reimplementation of the transformation chain.

3. The intermediate representation of variability models using HLCL does not provide full-solver independence. Indeed, the usage of multiple solvers in the HLCL is unpractical because it requires an extra transformation step to encode HLCL representations into solver-readable code, as shown in Figure 3.3. Thus, the inclusion of new solvers involves the definition and implementation of new transformation rules.

4. The intermediate representation of variability models using HLCL does not solve the issues regarding the portability and sharing of models between tools. Variability models initially described in different notations and later encoded into HLCL representations are useful for being analyzed in an integrated way in the same tool [DTS+14, MFTR+15]. However, the HLCL framework replicates the portability issues inside the intermediate representation layer as a consequence of the in-house definition and implementation of the HLCL. HLCL is a subset of constraint programming containing the operators and expressions employed to encode variability models. Therefore, a standard representation of constraint programming may be a better candidate to encode models as constraints in a solver-independent way.

The second iteration of the design cycle considered the results obtained in the exploration phase. The following paragraphs describe how the issues in the HLCL framework were addressed in the design, implementation and evaluation of the framework presented in this dissertation.

The design phase is the second iteration of the design cycle. This phase covered the design, implementation, and evaluation of the Coffee framework. This framework deals separately with the modeling concerns and analysis concerns as two parts of a single framework. The framework's design exploits the hypothesis developed in this thesis that introducing intermediate representations eases the interoperability between tools and the coupling in tool's implementations. Then, the second iteration encircles the following activities. Mind that the activities listed below were not performed sequentially.

Activity 4. Conceptual modeling the Coffee framework. The conceptual model of the framework was proposed

and iteratively refined during the *design phase*. Chapter 3 describes the conceptual model and presents its components.

Activity 5. Design, implementation, and evaluation of the High-Level Variability Language (HLVL) HLVL is the proposal of this thesis for a textual variability modeling language expressive and flexible enough to be considered as modeling language itself and as an intermediate language to exchange models and ease the portability of models and sharing issues between modeling tools. The design, implementation, and evaluation consisted of the following subactivities.

A.5. 1. *Definition of the syntax, semantics and characteristics for HLVL.* The decisions taken in the design of HLVL are consequences of results and lessons learned in the exploration phase. The language's design considered the modeling concepts gathered on the literature review, the criteria for ontological expressiveness, the characterization of textual variability languages, the different modeling paradigms, and the interchange with experts in the area. Chapter 4 details HLVL and its characteristics.

A.5. 2. *Implementation of the HLVL's platform.* Implementation of the editor and tools to describe variability models using the HLVL's syntax. The software components produced in this activity rely on Xtext, Xtend, and Java technologies. Chapter 5 presents the implementation of the HLVL's platform and its integration to the proposed framework.

A.5. 3. *Ontological evaluation of the expressiveness of the HLVL.* The final step in the definition of the modeling language was to perform an ontological evaluation. This evaluation aims to ensure the expressiveness of HLVL (*cf.* Chapter 5). Thus, the proposed language would be able to encode most variability modeling languages without expressiveness loss.

Activity 6. Design and evaluation of the analysis framework. After addressing the problem of representing variability models using an intermediate language without expressiveness loss, the next step was to provide a framework to analyze HLVL models. The following are the subactivities performed to define, implement, and evaluate the analysis framework.

A.6. 1. *Definition of the Operational Semantics of HLVL.* The general idea underlying operational semantics is to define the steps in the computation of the output from program phrases or code segments [Plo81, Läm18]. Thus, the operational semantics of HLVL is a collection of transformation rules applied to transform a model into a logic representation to perform satisfiability questions on solvers. Chapter 4 presents the operational semantics for each language construct considering different logic representations and solvers.

A.6. 2. *Demonstration of the feasibility of the context-aware analysis framework.* The analysis framework's definition was completed with the definition of the architecture and the implementation of a prototype. The accuracy of the solution was measured comparing the set of configurations obtained with an implementation of Coffee and the ones obtained with other management tools. Chapter 5 describes the particulars of the implementation and discusses the obtained results.

1.4 Summary of contributions

This thesis addresses the problems regarding the interoperability between variability management tools, the diversity among variability modeling languages, and the strong dependencies in the automated analysis of variability models. To solve these problems using intermediate languages, this thesis presents the following original contributions:

1. A systematic mapping study in the subject of variability management tools supported by constraint-based frameworks (Chapter 2). For this, a framework to analyze and compare the related literature is developed, consisting of multiple facets regarding the variability modeling concepts translated into constraints, the constraint systems used to encode variability models, the transformation rules to encode variability models as constraint satisfaction problems and the solvers supporting analysis tasks.

2. A theoretical framework to evaluate the expressiveness of variability languages from the ontological perspective. This evaluation framework is grounded on the theory of ontological expressiveness [WW93] and its application for variability modeling languages proposed by Asadi *et al.* [AGWH12]. Also, the evaluation framework relies on the the foundational ontology for variability included in the works of Reinhartz-Berger *et al.* [RBSW11] and the metrics for determining the criteria for completeness and clarity introduced by Recker *et al.* [RRIG09]. This framework was applied to evaluate the expressiveness of constraint programming for representing variability models (Chapter 3) and the modeling language proposed in this thesis.

3. The definition of a constraint-based transformation framework called Coffee (Chapter 3). The definition and implementation of the framework integrate the previous contributions, dealing with the variability concerns and the analysis concerns separately but considering those concerns as a part of a single proposal. Coffee is a four-layered framework containing a collection of methods and artefacts to encode variability models in HLVL, to encode HLVL models into constraint satisfaction problems, and to perform analysis tasks using the solver-independent infrastructure.

4. The design and formal definition of a textual variability modeling language called the High Level Variability language, HLVL for short (Chapter 4). The main goal of designing HLVL is to provide domain engineers with a rich textual language to model variability in academic and industrial contexts. The main characteristics of the language are its *expressiveness* from the ontological point of view and its *flexibility* to represent concepts present in variability languages with different modeling styles. This language, can be used as variability modeling language, and as intermediate representation of other variability languages. The general idea is that variability models can encoded in HLVL as an intermediate form and can be interpreted on other tools reducing the interoperability and sharing issues.

5. The definition of the operational semantics of HLVL in the form of inference rules considering the context of the transformation (Chapter 4). This operational semantics deals with the trade-off between expressiveness and analyzability by encoding HLVL models into satisfaction problems using two logical paradigms. Also, variability models are encoded in a second intermediate representation using a generic constraint representation provides a solver-independent solution. Thus, this proposal considers different expressiveness levels, transformation rules, solving paradigms, and solving tools.

1.4.1 Publications

This thesis extends and reuses contents of papers that are published, under review, or in preparation. Each chapter in this dissertation references the corresponding publication or manuscript. The following is the list of publications:

- Ángela Villota, Raúl Mazo, and Camille Salinesi. A CP-Based Framework for Product Line Engineering. Forum Jeunes Chercheurs du congrés INFORSID (2016). Actes du 8 e Forum Jeunes Chercheurs congrés INFORSID.
- Raul Mazo, Ángela Villota, Camille Salinesi and Daniel Diaz. MEDIC: Method to Diagnose Inconsistent Product Line Models using Constraint Graphs. To be submitted to the Requirements Engineering journal.
- Ángela Villota, Raúl Mazo, and Camille Salinesi. On the Ontological Expressiveness of the High-Level Constraint Language for Product Line Specification. In: Khendek F., Gotzhein R. (eds) System Analysis and Modeling. Languages, Methods, and Tools for Systems Engineering. SAM 2018. Lecture Notes in Computer Science, vol 11150. Springer, Cham. DOI: https://doi.org/10.1007/978-3-030-01042-3_4
- Ángela Villota, Raúl Mazo, and Camille Salinesi. 2019. The High-Level Variability Language: An Ontological Approach. In Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B (SPLC '19). Association for Computing Machinery, New York, NY, USA, 162-169. DOI: https://doi.org/10.1145/3307630.3342401.
- Ángela Villota, Raúl Mazo,Camille Salinesi and Daniel Diaz. Constraints in Software Product Line Engineering: a Classification Framework and Systematic Mapping Study. Manuscript in preparation.

1.4.2 Tools

Three different tools were developed during the Ph.D. These tools, are part of the artifacts developed at different stages of the research as reported in Section 1.3. Table 1.1presents the links to the tools' GitHub repositories.

Table 1.1: Summary of implemented tools

Tool	Available at
MEDIC	Personal Repository https://github.com/angievig/VARIAMOS, https://github.com/angievig/MEDIC_TESTS VariaMos Repository https://github.com/SPLA/VARIAMOS
HLVL Editor	Personal repository https://github.com/angievig/Coffee/tree/master/HLVL
Coffee Platform	Project's repository https://github.com/coffeeframework

MEDIC. The design, implementation, evaluation, report, and deployment of the Method to Diagnose Inconsistent Product Line Models using Constraint Graphs (MEDIC). This work was carried out during the first stages of the Ph.D. when I was developing a graphical representation for the High-Level Constraint Language. As a result of this early exploration, I included the constraint graphs as a notation to describe variability in the VariaMos tool suite. Additionally, I developed MEDIC, a diagnosis algorithm to detect inconsistent sets on a variability model specified as a constraint graph. The diagnosis algorithm was also integrated into the stand-alone version of the VariaMos tool suite. Moreover, the experimentation and benchmark related to the design and development of MEDIC are part of an upcoming journal paper.

HLVL Editor. This tool supports the grammar and the operational semantics for an early version of the High-Level Variability Language presented and discussed at the MODEVAR [VMS19]. This editor was implemented in java using the Xtext technology.

Coffee Platform. To support automated analysis associated to different variability languages, a microservices architecture of Coffee was designed and developed. In this microservices version of the framework, each level was implemented using Java and a REST API, then packed in a container to be deployed in the Docker swarm located at Icesi University in Cali. The design and implementation of the Coffee microservices platform is part of a project funded by Universidad Icesi with the colaboration of students from the i2t Lab.

1.5 Road Map of the Dissertation

This dissertation contains six chapters divided in three parts followed by an Appendix with auxiliary information.

Part 1, Motivation and Context. This part gathers the chapters delineating the research context, method, and the state of the research associated to this thesis.

Chapter 2. State of Research presents a systematic review and a classification framework of the existing research related to constraint-based frameworks supporting variability modeling and analysis. This chapter presents a summary of the variability modeling concepts, transformation rules to encode models into logic paradigms, and the solvers used in the automated analysis of variability models.

Part 2, Studies and Results. The second part of this dissertation presents three chapters containing the studies concerning the main focus of this thesis: expressiveness of variability modeling and analysis support to expressive variability modeling languages.

CH 3. From the Evaluation of the High-Level Constraint Framework to the Coffee Framework summarizes the results obtained in the evaluation of the HLCL framework performed during the exploration phase. This chapter focuses on the design, execution and results of the ontological analysis for constraints as variability modeling language. And more particularly, how the obtained results together with the experience obtained on the exploration phase contributed to the conceptual modeling of the Coffee framework. The chapter closes with an overview of the framework.

CH 4. Variability Modeling and Variability Analysis in Coffee presents the High-Level Variability Language HLVL its formal definition, operational semantics, and workflow from variability models to analysis process. First, Chapter 4 introduces the language, describes its characteristics, presents the formal syntax and semantics of the language. Second, the chapter explains how Coffee supports automated analysis using a three-step transformation framework to provide flexible, multi-language, and multi-solver support for automated analysis of variability models specified in HLVL. The transformation framework proposed in this chapter, is described in the form of inference rules in the definition of the operational semantics of HLVL.

Part 3, Results analysis, discussion and outlook. The last part of this dissertation further analyses and discuss the results.

CH 5. Evaluation, Discussion, and Outlook presents the evaluations used to demonstrate the expressiveness in the modeling language and the flexibility of the framework supporting variability analysis.

CH 6. Conclusions closes this dissertation with a summary of the results with respect to each research question, discusses the impact of this research, and presents the perspective for future work.

Appendix. Appendix A presents additional information produced in the execution of the systematic mapping study.

1.6 Summary

This chapter presented the context, the problem statement, addressed challenges, goals, and a contributions overview of this dissertation.

Chapter 2

State of Research

This chapter shares contents with the following papers:

- *Constraints in Software Product Line Engineering: a Classification Framework and Systematic Mapping Study.* Manuscript in preparation.
- *The High-Level Variability Language: an ontological approach* [VMS19]

This chapter presents a systematic literature study on constraint-based transformation frameworks supporting variability management tasks. The aim of this study is to understand the reach of constraint-based approaches in the life-cycle of variability intensive systems. The chapter is divided in four main sections. The first section presents the research method designed following the guidelines from Petersen *et al.* [PVK15]. The second section describes the classification framework. The third section presents the results and the answers to the research questions. To close, the fourth section presents the discussion and perspectives of the results.

2.1 Motivation

In Evidence-Based Software Engineering (EBSE) research methods, systematic mapping studies are defined methods to build a classification scheme and structure a field of interest [PFMM08]. As Petersen *et al.* emphasize, once the number of publications in a research area reflects a rapid increase, an analysis of existing works becomes important for a better understanding of that area and for identifying research gaps. In the domain of variability management there are several detailed literature reviews. However, most of them are centered on tools using feature-oriented languages. The most relevant literature reviews to this work are described as follows.

- Benavides *et al.* [BSRC10] and Galindo *et al.* [GBT+18]. In their paper, Benavides *et al.* presents (1) an overview of the different approaches supporting analysis tasks for feature models; (2) a set of transformation rules to encode feature-oriented models into logic expressions and constraint problems; and (3) a comprehensive catalog of analysis operations for feature-oriented models. This work was continued and extended in Galindo *et al.*'s work . This extension considered publications from 2010 and 2017 and considering other variability-intensive systems different than software product lines.
- Lopez-Herrejón *et al.*'s two systematic mapping studies about search-based methods [LHLE15] and testing [LHFRE15] on Software Product Lines. These works consider constraint approaches in the collection of

techniques for search-based methods and combinatorial interaction testing techniques for software product lines.

- Two literature reviews about tools for variability management [ACF14, MTS+14]. Alves *et al.*'s systematic literature provides an overview of the management tools that can be used as support the developing of software product lines, while Meinicke *et al.* present a literature review about the tools for supporting software product lines. These literature review contain an overview and a classification of tools for analyzing variability models and as a set of desirable characteristics for product variability management tools.
- The literature reviews by Eichelberger & Schmidt's and ter Beek *et al.* [EKS13, ES15, tBSE19]. In [EKS13], Eichelberger & Schmidt systematically analyze feature-oriented languages and discussing the trade-off between the expressiveness of the modeling language and the complexity of the logic paradigm needed for represent such models. Also, Eichelberger & Schmidt present in [ES15] a survey on textual variability modeling languages. This survey presents a classification scheme to describe the capabilities of textual variability modeling languages. Ter Beek *et al.* in [tBSE19] updated the later review consolidating the classification scheme and including three textual languages.

The mapping study developed in this chapter aims to analyze and categorize the peer-reviewed literature, their contributions, and corresponding applications. Also, this mapping study covers literature with variability described in different modeling paradigms. Consequently, this chapter provides a clear view on (1) which variability concepts are modeled with constraints; (2) how variability models are transformed into constraints; and (3) how variability management is supported by solvers and variability-management tools. The outcomes of this systematic mapping study are:

- Identifying the variability modeling concepts expressed as constraints.
- Characterizing the constraint-based software tools, and solvers used for variability management.
- Providing the practitioners with guidelines about the tools, techniques, and applications of constraint-based approaches for variability management.

2.2 Research Method

The design of this mapping study follows the guidelines and methods proposed by Petersen *et al.* [PVK15] and the recommendations for building systematic literature reviews proposed by Kitchenham *et al.* in [KPP+02]. Figure 2.1 depicts the five step process to conduct a systematic mapping study [PFMM08]. The following subsections detail how each of the five steps were applied in this study, most artifacts, such as tables and list are gathered in Appendix A.

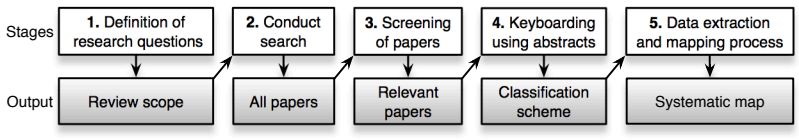


Figure 2.1: Systematic mapping study process [PVK15]

2.2.1 Research Questions and Scope

The following research questions guide the current literature review[1]:

SMS.Q1 What variability concepts are modeled as constraints (described directly as constraints or transformed into constraints)?
This questions drives the literature review to elicitate the variability modeling requirements. This question was divided into the following three sub-questions:

- What is/are the paradigm(s) used to describe variability?
- What are the variability's concepts used by the variability language used in the specification?
- Does the model specification include new constraints (different from the provides by the variability notation)? What is the purpose of the inclusion of the new constraints?

SMS.Q2 What types of constraint systems (*i.e.,* variables, domains, and constraints) are used to encode variability models and for analysis purposes?
This question drives the literature review to understand the diversity in the transformation framework supporting analysis tasks. This question was also divided into two sub-questions as follows.

- Which rules were applied to transform variability models into constraints?
- What are the types of the constraints and domains used in the transformation?

SMS.Q3 What are the characteristics of the solvers supporting variability management?
The support provided by solvers is relevant to delineate the nature of the solving needs in the area. **SMS.Q3** produces the following subquestions:

- Do publications report the use of a solver?
- Which solver?
- What is the solver's paradigm?
- How is the solver implemented? (as a library, language, stand-alone tool)

SMS.Q4 What are the characteristics of the variability-management tools?
This question is relevant to characterize constraint-based tools frequently for variability management. **SMS.Q4** produces the following subquestions:

- Does the publication relates a constraint-based software-tool, or presents the implementation of a software-tool?
- What is the name of the tool?
- What is the type of tool?
- What is the software tool used for?

[1]These questions are coded with the prefix SMS to indicate that they are in addition to the main research questions and are addressed only in this chapter through the study of literature

The research questions do not address variability modeling languages as the works in Eichelberger & Schmidt [ES15] and Ter Beek, Schmidt, and Eichelberger [tBSE19] already define a classification scheme to describe the capabilities of variability modeling languages. Moreover, these literature reviews represent a basis for discussing future options and challenges towards the design of variability modeling languages. Consequently, the evaluation of the modeling language proposed in this thesis will consider classification scheme in [ES15, tBSE19] to compare the language against other proposals.

To conclude, the collection of questions and subquestions the systematic review process to analyze and classify the research involving the constraint-based approaches in the life-cycle of variability intensive systems. Additionally, the 13 subquestions conform the extraction questionnaire (See Appendix A).

2.2.2 Conduct Search for Primary Studies

The search considered the concordance of search strings in the title, abstract, and keywords of the studies published since 1997. This year was chosen because is the year of the publication of the report for the first Product Line Practice Workshop organized by the Software Engineering Institute (SEI).

Defining Search Terms

Based on the objective of finding publications discussing the application of constraint-based approaches in the life-cycle of variability intensive systems, the search strings were built using three sets of terms. Table A.1 in Appendix A presents the list of terms. Two search strings resulted from combining the sets of terms using the logic operator AND.

1. The string produced from combining the first and second set of terms.
2. The string produced from combining the first and third set of terms.

The first group of terms in Table A.1 contains the set of terms related to variability management (*i.e.,* modeling and analysis). These terms were selected considering the terms used in secondary studies in the variability management domain [ACF14, MTS+14, MP14]. The second group contains keywords regarding constraint-based approaches. These terms were selected based on constraint programming, constraint logic programming and satisfiability solvers. Finally, the third group of terms contains a list of solvers validated by experts.

Search Strategy

The search and selection of the set of relevant studies was conducted in three steps:

1. **Manual Search.** The search of relevant publications started by conducting a manual search over a collection of conferences, workshops, and journals related to software engineering, software product line, and constraint-based approaches. This manual search was useful for refining the search strings and testing the inclusion/exclusion criteria. Table A.2 in Appendix A presents the list of selected venues. This list, includes the venues from secondary studies in the SPLE domain [BSRC09, BSRC10, MP14, LC13] and others relevant for constraint-based approaches.

2. **Automated Search.** The second step in the search strategy was an automated search introducing the search strings into the search engines provided by Scopus and Web of Science (WoS). The selection of Scopus and WoS instead of every particular database is justified because the search service provided by Scopus indexes journals and proceedings of conferences residing in digital libraries in computer science (i.e.: ACM Digital Library, IEEE-Xplore, Science Direct, Springer). Also, WoS allows the automated search over journals and specialized publications in the computer science area.

3. **Snowball Search.** The search strategy has a complementary step with a snowball search. The snowball search consisted of three activities (1) scanning the references in each paper (2) searching for related papers using Google Scholar (3) examining the publications listed in DBLP for a set of frequent Authors. This study consider a frequent author to each author whose name appears in five or more publications in the collection of relevant papers (no matter the order of the names in the publication). To avoid duplicates and redundancies, the snowball search considered the set of publications after applying the exclusion/inclusion criteria to the papers obtained in previous steps.

2.2.3 Screening Papers - Inclusion/Exclusion Criteria

The screening of the papers at this stage allowed the elimination of publications complying with the search strings but irrelevant to the systematic mapping study. Therefore, this stage requires an inspection of the title, abstract, introduction, conclusion and in special cases, the entire paper until no doubts were left about its selection. The following are the inclusion/exclusion criteria:

- The systematic mapping study includes publications with clear applications of constraint-based approaches in the variability management context.
- The systematic mapping study excludes publications written in a language different to English and written in the form of editorial, abstract, keynotes, and posters.
- The study includes publications of the same author with similar content. Publications of the same author with similar contents are considered as independent studies and are relevant to understand the evolution of the authors' work.
- Secondary studies are gathered but not included in the list of selected documents. Secondary studies are recorded to be used as references to evaluate the relevance of this systematic mapping study and to analyze the related work.

As a result, this staged produced a collection of 137 relevant papers, out of which 87 belong to conference proceedings, 28 are journal articles, 22 come from workshop proceedings, and ten are book chapters.

2.2.4 Data Extraction and Mapping Study Process

The data extraction of each paper followed a three-step process. This process was evaluated and tuned through the conduction of a pilot review. This pilot included twenty papers randomly selected. As a result, the instruments were adjusted. Section A.3 in Appendix A presents the extraction instruments.

In the first step, the extraction of data is guided by answering the data-extraction questionnaire (*cf.* Section A.3, Appendix A). The questionnaire contains a list of short-questions answered by assigning a predefined label or category in the classification framework. This process caused the iterative extension of the classification framework. The following section provides a detailed description of the classification framework and its contributions for this chapter.

The objective of the second and third step was to gather bibliometric data. The second step required to gather information about the authors, such as, the affiliation and countries. The third step was useful to ensure the completion of bibliographic information in bibtex format.

2.2.5 Threats to Validity

Threats to validity of this mapping study are analyzed according to the following aspects: descriptive validity, theoretical validity, and reliability accordingly to Petersen *et al.* definitions in [PVK15].

Descriptive validity refers to the accuracy and objectivity of the information gathered during the review stage. To reduce the potential bias while recording the data, measures were taken during the review process and regarding the information recording instruments. First, the protocol of this study includes a process to review and extract the information of each paper in a systematically way. This process contains a series of steps to ensure the recording of the relevant information and that all questions obtain an answer. Additionally, the review process was assessed and adjusted during a pilot review. Second, the research questions were divided into an extended set of questions where the answer is one or more elements in a predefined set. The reader may observe that each facet in the classification framework corresponds to one of the extended questions (*cf.* Table 2.1). Many iterations over the classification framework allowed us to adjust attributes and domains when needed. This chapter presents the final version of the resulting framework.

Theoretical validity is determined by the ability to capture what the study intends to capture [PVK15]. This mapping study ensures the theoretical validity by performing a suitable selection of primary studies. To that end, measures were taken during different stages such as the search strings construction, search, and filtering of the papers following a predefined set of inclusion/exclusion criteria. To address the threats regarding the selection of search strings, this study presents a carefully selected collection of search terms from variability management and constraints domains. From the variability management point of view, the selection included search terms based on previous literature reviews, such as, [BSRC09, BSRC10, MP14, LC13]. Furthermore, the constraint programming terms were selected based on keywords suggested by domain experts.

Another threat to validity comes from how the search for primary sources was carried out. To address the threats regarding the search stage, this study proposes a three-level search including manual search, automated search, and snowballing search. The manual search covered the most relevant venues in software product lines and constraints domains. Also, the automated search with the help of Scopus and WoS search engines covered the conferences and journals from different digital libraries, and specialized publications in the computer science area. Finally, the collection of selected publications included the publications obtained by snowball searching and a manual search over DBLP profiles of the most frequent authors in the set of publications.

To decrease the threats regarding the filtering process, the study incorporated the inclusion/exclusion criteria

in the protocol and the reviewing process. The integration of inclusion/exclusion criteria as a measure ensures the consistent filtering of all publications. Nevertheless, the first filtering was based mainly on the occurrence of search strings on the titles and keywords, as well as a succinct read of the abstracts. In consequence, some relevant papers related to the topic might have not been included.

Reliability refers to the accuracy of the conclusions drawn in relation to the data collected [PVK15].This study reduced the possible bias in the interpretation of data, by conducting a series of assessment meetings. These meetings allowed the main author of the thesis and her collaborators to evaluate and discuss the results and conclusions. After all authors agree, the results and conclusions were included in the report.

2.3 Classification Framework

The classification framework is a hierarchical structure inspired in the works of Rolland *et al.* [RAC+98] and Livari [Iiv89]. The hierarchical structure of the framework contains three levels: views, facets, and categories. Views represent subjects, they associate the facets to a specific focus in the classification. A facet represents a classification criterion that is related to a set of categories.

Table 2.1 summarizes the classification framework. It contains three views: modeling, transformation, and support. Their brief introduction is presented below:

- The *modeling* view relates the facets associated to syntax and semantics of variability (i.g. concepts, aspects, tasks, processes).
- The *transformation* view gathers the facets related to the constraint systems and rules employed in the transformation of variability models into constraints.
- The *support* view is used in the classification of the constraint-based software tools employed to support variability management.

The next subsection provides a detailed description of the set of facets in the classification.

2.3.1 Modeling-centered Facets

The facets and categories in the expressiveness view answer the first research question (**SMS.Q1**). Three facets compose the expressiveness view: the modeling paradigm, the modeling concepts, and the variability enhance facets. The modeling concepts facet contains the glossary of variability modeling concepts obtained in this literature review and published in the paper [VMS19].

Modeling Paradigm

The modeling paradigm is the approach employed to capture variability in the form of models, programs or code. The modeling paradigm defines the semantics of the *variability unit*, the *structure* of the model, and the *configuration process*. Variability is documented in one of the following paradigms:

1. Feature-oriented models, where the variability unit are features. Feature-oriented models comprehend numerous languages derived from FODA [KCH+90].

Table 2.1: Classification scheme with multiple facets and their associated categories.

View	Facet	Categories
Variability modeling	Modeling paradigm	feature-oriented, variation point-oriented, decision-oriented, constraint-oriented, goal-oriented, UML-based.
	Modeling concepts variability units	Type-Boolean, Type-non-Boolean, single-valuation, multiple-valuations, single-instance, multiple-instances, attributes, attached-info.
	Modeling concepts variability relationships	Commonality, decomposition one-to-one, decomposition one-to-many, simple, propositional, first-order, relational, arithmetic.
	Variability-enhancement	Visibility, conditioned inclusion, conditioned exclusion, quantified implication, global, traceability, soft-constraints, NFRQ, time.
Transformation	Constraint Systems	Integers, Booleans, Real numbers, symbolic, special variables.
	Transformation Rules	R_i, $1 \leq i \leq 23$
Support	Solver implementation	Programming language, library, tool.
	Solving paradigm	SAT, BDD, SMT, ILP, MILP, CSP, CLP, extended.
	Software-tool - Type	language, algorithm, tool, or variability-management tool
	Software-tool - evolution	Use, extend. propose.
	Software-tool - implementation	Prototype, Plugin, Stand-Alone, Web-Application
	Software-tool - function	Analysis, verification, configuration, testing, derivation, simulation, synthesis

2. Variation point-oriented models, where variation points and variants are the variability units. Variation point-oriented models document variability in a different and dedicated model separating the variability documentation from the documentation of the software development artifacts [PBvdL05].
3. Decision-oriented models, that use the concept of decisions and dependencies among decisions to describe variability in the form of decision models or tables [DGR11].
4. Constraint-oriented models, that use constraint-expressions directly to describe variability (not transformed).
5. Goal-oriented variability models, use goals as variability unit to represent variability for self-adaptive software systems [SMD+12]. These models define the relationships between a system and its environment to produce configurations at runtime.
6. UML-based variability models, include variability information within the documentation of the software development artifacts using UML or extending the UML syntax [LDSSHC15b].

Modeling Concepts

The modeling concepts in variability languages enable the modeler to answer two questions about the variability-intensive system to be modeled: *what does vary?* and *how does it vary?* [PBvdL05]. Modeling concepts in variability languages are tools (1) to identify and document the variable items in a system; (2) to identify the set of possible options or variants associated to variable items in the system; (3) to identify the rules for determining how items can be combined into new configurations; and finally (4) to produce variability models. Modeling concepts are grouped in the facets related to *variability units* and related to the *variability relations*.

Variability Units Facet. Variability units are the key concept used to model variability in a language [CGR+12]. They represent variable items in a system or domain, that is, those aspects that must be chosen by the customer or engineer in a configuration process. For example, *features* are the variability units in feature models, *decisions* in decision models, and *variation points*, *variants* are the units in OVM models. The following are the categories for the variability units facet:

1. **Regarding the type.** The type of a variability unit is defined by the number of variants it is associated with. There are two categories regarding type: Boolean and non-Boolean. Variability units are **Boolean** when they are associated with exactly two variants: *{selected, unselected}*, as in feature models. **Non-Boolean** variability units have more than two variants. For example, numeric features in feature-oriented models, or decisions in decision models.

2. **Regarding the number of valuations.** The number of valuations is the amount of variants a variable item is assigned in the configuration process. There are two categories regarding number of valuations: single and multiple. **Single** valuation units are associated with only one value in the set of variants. For instance, features are linked either to the selected or unselected value, but not both. **Multiple** valuation units can be associated to more than one variant after the configuration process. For instance, industrial languages, such as Gears, allows the units to be declared as sets, or records that can be assigned to more than one value [Kru07].

3. **Regarding the multiplicity.** Multiplicity represents the number of instances a unit may appear in a configuration. Some variability languages allow annotations affecting the variability units using multiplicities in a UML style. There are two categories regarding the multiplicity: single-instance and multiple-instance. **Single-instance** units have a unique instance in the configuration. **Multiple-instance** units may have more than one instance.

4. **Attributes.** Attributes are labels linked to variability units. They are not variability units by themselves, as they represent properties or particular characteristics of variable items in a system. There is no consensus on a notation to define attributes. Most proposals agree that their definition should include a type, a name, a domain, and optionally a *value*. [BSRC10].

5. **Attached information.** Attached information does not modify the semantic of the model. However, some languages allow the introduction of comments or labels attached to the variability units to provide extra information.

Variability Relationships Facet. The variability relationships determine the rules to select and recombine items into new products. Variability relations are often presented as dependencies or constraints and are usually denoted graphically (i.e., using arrows) or textually (i.e., logic formulas, OCL). Variability models contain more than one variability relationships. These relations can be classified as follows:

- **Commonality** relationships are rules for defining items that always appear in any configuration. This rule is implicit in some languages, *e.g.,* root feature in feature models, or non-existent as in decision models.
- **Hierarchy-Decomposition** relationships are n-ary relations where one item plays the role of parent, and the other are the children. This parent-child relation imposes a constraint in the configuration because no child can be part in a configuration without the inclusion of its parent. There are two types of decompositions:
  1. *On-to-one* decompositions relate pairs of items. There are two types of decompositions: **mandatory** and **optional**. In mandatory decompositions, the child is included in all products in which its parent appears. Instead, in optional decompositions, the child can be optionally included in all products in which its parent appears.
  2. *One-to-many* decompositions relate one parent and a group of children. This relation restricts the minimum and the maximum number of children that may be included in a configuration when their parent is selected.
- **Constraint expressions** are used to include complex rules between variable items in a model. These constraint expressions are often used to specify extra-functional information or to include contextual rules [KOD13]. Regarding the operators in the expressions there are five categories of constraint expressions as follows:
  1. Simple constraints, where the language provides the constructs to define the inclusion/exclusion of items *e.g.,* requires and excludes.
  2. Propositional constraints, where the language supports expressions in propositional logic.
  3. First-order constraints, that are included to support expressions in first-order logic.
  4. Relational constraints, that are included to support relational operations.
  5. Arithmetic constraints, where the language includes arithmetic operators usually to introduce calculations containing attributes and non-Boolean variability units [KOD10b].

Variability-enhance Facet

A limited number of primary studies in the review included new constructs or constraints to enhance variability descriptions. This facet presents seven categories grouped by constructs and constraints.

Constructs. The following are the categories of variability relationships expressed with language constructs:

1. Visibility constraints, that are relationships conditioning the availability of other variability items. Visibility relations are considered a type of hierarchical relations because they are used to compartmentalize items,

as in different views, e.g., for different stakeholders [CGR+12]. Visibility relations are a common construct in decision-oriented languages such as Dopler [DGR11].

2. Conditional inclusion/exclusion relationships, that are rules for restricting the inclusion/exclusion of variable items given a condition. Languages such as CO-OVM[DTS+14], and extended-feature models in [KOD13] allow the usage of complex expressions to condition the inclusion/exclusion of variable items using logic expressions.

3. Quantified requires, that is a relationship introduced by Quinton *et al.* [QRD13] and revisited in other studies. This relationship conditions the number of instances of a variable item required to include another.

Constraints. The following are the classification of the types of constraints included in variability models to enhance the variability specification:

1. Global constraints are used for incorporating constraints among attributes and features, regardless the tree structure of variability models [KOD10a].

2. Traceability constraints are relations between features and sets of artifacts implementing the corresponding features [MRM+12].

3. Soft-constraints are relations between elements and values, which satisfaction is desirable rather than required. They represent context rules in self-adaptive systems modeled as dynamic software product lines [SMD+12], [MFTR+15].

4. Optimization constraints are constraints related to an objective function that should be maximized or minimized (or both in multi-objective optimization problems).

5. Non-Functional Requirement Constraints (NFRQ) are expressions used in domain analysis, for specifying variability requirements when extra-functional information is included.

6. Time constraints are constraints introduced to declare relationships that affect the order or time a variability item is configured.

2.3.2 Transformation-centered Facets

The transformation centered facets contains the categories to answer the second research question (**SMS.Q2**). Two facets compose the transformation view: the constraint system and the transformation rules facets. The following subsections present the facets and their categories.

Constraint System

A constraint system specifies the types of variables and kinds of constraints a solver can handle in terms of sets, functions, and predicates [SR90]. The constraint system facet classifies the publication regarding the type of variables and constraints used to formulate constraint problems. Variability models are transformed into constraint satisfaction problems using the following constraint systems:

1. Constraint systems over integers where the variables may range in a set of positive integers also known as finite domains. These constraint systems use arithmetic and relational operators and global constraints provided by the solver.
2. Constraint systems over Booleans where the variables in the constraint problem may range over *Booleans* in the set of values is $\{True, False\}$ or equivalently, in the integer domain $\{0, 1\}$. Boolean variables have constraints expressed using logic formulas.
3. Constraint systems with variables ranging over Real numbers. Solvers with the capability to deal with variables ranging over Real numbers require special algorithms or techniques, such as interval arithmetic, or linear programming algorithms.
4. Constraint systems where the variables may contain non-numeric values ranging over *symbolic domains*. The constraints in these systems are often provided by solver's special packages or libraries.
5. Constraint systems with special variables, such as sets and records. Similarly to symbolic domains, the constraints for sets and records are provided by special packages or libraries.

Transformation Rules

As recurrently explained above, the encoding of the variability model as a constraint programs is a requisite to perform variability analysis. The categories in the transformation rules facet corresponds to the list of documents containing new proposals or extensions of transformation rules. Table 2.2 presents the list of publications, each with its first author, the publication year and the title of the publication.

Table 2.2: List of publications proposing or extending transformation rules.

Id	Reference	First author	Year	Title
R1	[SS02]	M. H. Sørensen	2002	From Type Inference to Configuration.
R2	[Man02]	M. Mannion	2002	Using First-Order Logic for Product Line Model Validation.
R3	[VK02]	A. van Deursen	2002	Domain-specific Language Design Requires Features Descriptions.
R4	[ZZM04]	W. Zhang	2004	A Propositional Logic-Based Method for Verification of Feature Models.
R5	[BTRC05a]	D. Benavides	2005	Automated Reasoning on Feature Models.
R6	[Bat05]	D. Batory	2005	Feature Models, Grammars, and Propositional Formulas.
R7	[SZFW05]	J. Sun	2005	Formal Semantics and Verification for Feature Modeling.
R8	[BSTRC06a]	D. Benavides	2006	A First Step Towards a Framework for the Automated Analysis of Feature Models.
R9	[BSTRC06b]	D. Benavides	2006	Using Java CSP Solvers in the Automated Analyses of Feature Models.
R10	[CW07]	K. Czarnecki	2007	Feature Diagrams and Logics: There and Back Again.
R11	[DS07]	O. Djebbi	2007	RED-PL, a Method for Deriving Product Requirements from a Software Product Line Requirements Model.

Continue on the next page

Table 2.2: Transformation rules and supported concepts. (cont.).

Id	Reference	First author	Year	Title
R12	[DS08]	O. Djebbi	2008	Towards an Automatic PL Requirements Configuration through Constraints Reasoning.
R13	[EPAH08]	O. A. Elfaki	2008	Knowledge Based Method to Validate Feature Models.
R14	[FO09]	R. Finkel	2009	Reasoning About Conditional Constraint Specifications.
R15	[WNS09]	L. Wang	2009	Constraint Satisfaction Approach on Product Configuration with Cost Estimation.
R16	[KOD10b]	A. Karataş	2010	Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains.
R17	[RFBC10]	F. Roos-Frantz	2010	Automated Analysis of Orthogonal Variability Models using Constraint Programming.
R18	[SMDD10]	C. Salinesi	2010	Using Integer Constraint Solving in Reuse Based Requirements Engineering.
R19	[CBH11]	A. Classen	2011	A Text-based Approach to Feature Modelling: Syntax and Semantics of TVL.
R20	[DFH11]	D. Dhungana	2011	Configuration of Cardinality-based Feature Models Using Generative Constraint Satisfaction.
R22	[MGH+11]	R. Mazo	2011	Using Constraint Programming to Verify DOPLER Variability Model.
R23	[SMD+12]	P. Sawyer	2012	Constraint Programming as Means to Manage Configurations in Self-Adaptive Systems.
R24	[SHTB07]	P. Schobbens	2007	Generic Semantics of Feature Diagrams.
R25	[BSTRC07]	D. Benavides	2007	FAMA: Tooling a framework for the Automated Analysis of Feature Models.
R26	[TMV+16]	A. Tidstam	2016	Formulating constraint satisfaction problems for the inspection of configuration rules.
R27	[WLS+16]	M. Weckesser	2016	Mind the gap! automated anomaly detection for potentially unbounded cardinality-based feature models.
R28	[MOP+19]	D. Munoz	2019	Uniform Random Sampling Product Configurations of Feature Models That Have Numerical Features.

2.3.3 Support-centered Facets

The Support view gathers the facets and categories to answer the third and fourth research questions (**SMS.Q3**). Three facets compose the support view: the *implementation approach* facet, the *solving paradigm* facet and the *software tool* facet. The next subsections will expand the description of the facets and its categories.

Implementation approach

This facet drives the classification of primary studies regarding the approach used for implementing the solver. There are three categories to classify a solver regarding the implementation approach:

1. Solvers provide implementations using programming languages that include the notion of constraint and solving techniques. These programming languages include a solver and provide the APIs for calling the procedures and queries to the solver as native constructs in the language.
2. Solvers may be implemented as libraries provided by languages where the constraint satisfaction is not one of the main concepts, for instance, there is the CHOCO library for Java and GECODE for C++.
3. Solvers can be implemented as software-tools. In this sense the solver is an external black-box that communicates with the principal program by a defined interface. For instance, solvers implemented as tools such as Yices, PicoSAT, and Z3.

Solving paradigm.

This facet drives the classification to determine he paradigm employed for modeling and solve the constraint satisfaction problems. The solving paradigm is related to the approach to encode variability models into constraint satisfaction problems for analysis purposes. The following are the four categories in the solving paradigm facet:

1. Solvers specially designed to solve propositional *Satisfiability* Problems (SAT). Solving SAT problems is to decide if there is a truth assignment under which a given propositional formula (written in conjunctive normal form) evaluates to true. The SAT problem is formulated as follows (from [CLRS09]): an instance of SAT is a boolean formula f composed of:
   (a) n boolean variables: $x_1, x_2, \ldots, x_n$;
   (b) m boolean connectives: any boolean function with one or two inputs and one output, such as $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT), $\rightarrow$ (implication), $\leftrightarrow$ (if and only if); and
   (c) parentheses.

   A truth assignment for a boolean formula f is a set of values for the variables of f. A satisfying assignment is a truth assignment that causes a formula f to evaluate to $True$. A formula with a satisfying assignment is a satisfiable formula. The satisfiability problem asks whether a given boolean formula is satisfiable: $SAT = \{\langle f \rangle : f \; is \; a \; satisfiable \; boolean \; formula\}$.
2. Solvers for *Binary Decision Diagrams (BDD)*, these solvers also solve the satisfiability problem. The particular characteristic of BDD solvers is that the input is a logic formula represented as a data structure (binary decision diagram). BDD solvers provide functionalities as determine satisfiability and obtain the number of possible solutions.
3. *Satisfiability Modulo Theory (SMT)* solvers solve a generalized form of SAT problems. SMT solvers combine propositional logic with domain-specific reasoning. Therefore, input formulas are evaluated with respect to combinations of theories such as arithmetic, uninterpreted functions, data structures, etc. In general, SMT solvers determine if a ground first order formula is satisfiable regarding a background theory incrementing the expressiveness by including variables ranging in different domains than Booleans.

4. Solvers for *Integer Linear Programming (ILP)* and *Mixed Integer Linear Programming (MILP)*. These ILP and MILP are common approaches to solve optimization constraint problems with Real and Integer variables [BDRG10, Hei13]. The goal for solving a linear-programming problem is to optimize a linear function subject to a set of linear inequalities. Formally, from [CLRS09], given a set of real numbers $a_1, a_2, \ldots, a_n$ and a set of variables $x_1, x_2, \ldots, x_n$, we define a linear function f on those variables by $f(x_1, x_2, \ldots, x_n) = a_1x_1 + a_2x_2 + \cdots + a_nx_n = \sum_{j=1}^{n}(a_jx_j)$. If b is a real number and f is a linear function, then the equation $f(x_1, x_2, \ldots, x_n) \leq b$ and $f(x_1, x_2, \ldots, x_n) \geq b$ are linear inequalities, also called linear constraints. Therefore, a linear-programming problem is the problem of either minimizing or maximizing a linear function subject to a finite set of linear constraints.

5. Solvers for *Constraint Satisfaction Problem* (CSP). CSP solvers combine inference and search techniques to find one or more solutions to constraint satisfaction problems. Formally, from [RvBW06], given a set of variables $X = \{x_1, \ldots, x_n\}$ and a set of domains $\mathcal{D} = \{D_1, \ldots, D_n\}$, a CSP P is defined as a triple $\langle X, \mathcal{D}, \mathcal{C}\rangle$ where $\mathcal{C}$ is a set of constraints on X. Each constraint c in C is a relation defined on a sequence of k variables $X(c) = (x_{i1}, \ldots, x_{ik})$. Constraints can be specified extensionally by the list of its satisfying tuples, or intensionally by a formula that is the characteristic function of the constraint.

6. Solvers for *Constraint Logic Programming (CLP)*. CLP is an approach used to solve constraint satisfaction problems combining two declarative paradigms: constraint solving and logic programming [RvBW06]. Then, CLP solvers includes the rules, predicates, unification and resolution from the logic programming paradigm within the propagation and search techniques from CSP solvers. The mixture of paradigms in CLP provides powerful tools to the modeling and solving of constraint problems.

7. Extended solvers, gathers the solvers for not so frequently used approaches such as *Answer Set Programming (ASP-SAT)* and *Quantified SAT (QSAT)*, among others. Answer set programming solvers are used to specify variability in a conditional fashion and later configure products with the help of Alloy [FO09]. QSAT solvers are employed to solve satisfiability problems with quantified formulas [MRM+12].

Software tool

This facet guides the classification of other software-tools (different than solvers) used for supporting variability management. In this section a software tool is defined as a software element in the form of compilable or executable code or application employed in the development, repair, or management of a variability intensive system. The following are the categories in the software-tool facet:

Regarding the evolution of the software tool, a publication in the review may belong to one of the following categories:

1. *Use*, when the work uses a constraint-based software tool for variability management that is not the main contribution in the publication.

2. *Extend*, when the publication includes the contribution as an extension of an existent software tool.

3. *Propose*, when the publication introduces a software tool.

Regarding the type of software a software-tool reported in a publication belongs to one of the following categories:

1. Variability-management tool.
2. Modeling language.

Regarding its implementation a software-tool in a publication may be classified as:

1. A *prototype*, or a tool that serves as a feasibility test to assess the practicality of the proposal in the publication.
2. *Plugin*, that is a software solution presented as a piece or constituent of other development tools, such as Integrated Development Languages (IDE).
3. *Stand-alone* tools or programs that can work offline, *i.e.,* does not necessarily require network connection to function.
4. *Web applications* or tools that require a browser and internet connection to work properly.

Regarding the functionality provided by the tool. The categories are the following:

1. Variability modeling when the software tool provides a mechanism to document variability either graphically or textually.
2. Analysis when the software tool provides any of the analysis operations.
3. Verification is a particular case of analysis. Verification is the evaluation to determine if a product, service, or system complies with a regulation, requirement, specification, or imposed condition [PMI13].
4. Product configuration is the activity of solving the software product line model and searching the set of components that meet the constraints in the variability models and, the customers' requirements.
5. Testing a variability intensive system is a process that solves one of two problems: (1) test case generation [UGKB08] or (2) product generation [PSK+10].
6. Derivation is the complete process of constructing a product from product family software assets.
7. Simulation is a process linked to dynamic software product lines or self-adaptive systems where the tool validates the planned reconfiguration and includes the valid configurations in a feedback-loop [MFTR+15].
8. Synthesis is the process to obtain a feature model from a specification in the form of logic formulas [CW07].

2.4 Classification and Mapping

This section presents the classification and mapping of the publications gathered in the search stage with respect to the classification framework described in Section 2.3. The search stage produced a collection of 137 relevant publications. The classification included the recording of the bibliographic information for each document summarized in Appendix A.

2.4.1 What variability concepts are modeled as constraints?

This section presents the mapping and classification of publications regarding the modeling-centered facets in the framework, question **SMS.Q1**, and its derived subquestions. However, the first result of this classification is the glossary of modeling concepts described in in Section 2.3.1 as part of the classification framework. The two subsections below answer the questions regarding the modeling paradigms and the extension of variability languages with new constructs and constraint expressions.

Modeling paradigms

Table 2.3 shows the classification of the primary sources regarding the modeling paradigm used to describe variability. The rows in the table represent one of the categories in the modeling paradigm facet described in Section 2.3.1. Each row contains the list of publications and the number of documents using a paradigm. Not surprisingly, *feature-oriented* models are the most frequent paradigm with 119 publications. The second most frequent paradigm is *constraint-oriented* models with 30 publications, followed by *variation point* oriented models with 14 publications. The least used methods are *decision-oriented models*, *UML-based* models, and *goal-oriented* methods. These results confirm that *feature-oriented* methods are the most used paradigm to represent variability. Note that the sum of the amounts in the third column in Table 2.3 is greater than 137. The latter is because a publication can report more than one paradigm. For instance, consider the work of Munoz-Fernandez *et al.* [MFTR+15] using different modeling paradigms to represent variability.

Table 2.3: Classification of publications regarding the modeling paradigm.

Method	Publications	#
Constraint-based	[SS02], [FO09], [WN09], [SDD+09], [WNS09], [MSD11], [SMD+11b], [SM12], [MSD12a], [MSD+12b], [tBLP13], [ZKY+14], [WZZ15], [OGRT15], [tBLLLV15a], [tBLLLV15b], [tBLLLV16], [GMS15], [LDSSH15], [LDSSHC15a], [LDSSHC15b], [TMV+16], [SYP01], [VK02][MPH+07], [EPAH08], [EPAH09a], [EPAH09b], [EFV+13], [ASS+19]	30
UML-based	[GMS15], [LDSSH15], [LDSSHC15a], [LDSSHC15b]	4
Decision-based	[MGH+11], [NBE12], [DFH11], [DGR11]	4
Variation point-based	[MPH+07], [EPAH08], [EPAH09a], [EPAH09b], [RFBC10], [SMDD10], [SMD+11b], [SM12], [RFBRC+12], [EFV+13], [MDSD14], [Kru07], [SKES18], [DTS+14]	14

Continue on the next page

Table 2.3: Classification of publications regarding the modeling paradigm. (cont.).

Method	Publications	#
Feature-based	[SYP01], [ZZM04], [Bat05], [BTRC05a], [BTRC05b], [CK05], [SZFW05], [BSTRC06a], [BSTRC06b], [ZMZ06], [TBRC06], [BSTRC07], [CW07], [DS07], [DSD07], [MPH+07], [TBKC07], [WSWN07], [WSC+07], [DS08], [MWCC08], [EPAH08], [Seg08], [TBD+08], [UGKB08], [WSWN08], [WSB+08], [ZYZJ08], [CHH09], [MBC09], [MWC09], [EPAH09a], [EPAH09b], [SRM09], [STJ09], [TBK09], [TC09], [WZZ09], [WDSB09], [YZZM09], [BDRG10], [KOD10a], [KOD10b], [PSK+10], [SMD10], [SMDD10], [SK10], [SGW10], [UKB10], [Wan10], [WBS+10], [ACLF11], [CBH11], [DFH11], [HBG11], [MSDL11], [MLHS+11] [MCHB11], [MHG+11], [PLP11], [SMD+11b], [TBG13], [ZZM11], [ACSW12], [GRF+12], [MRM+12], [PRM+12], [SM12], [XHSC12], [ACL13], [BDA+13], [BTCS13], [FBGR13], [Hei13], [HPP+13], [KOD13], [LHCF+13], [MGSH13], [EFV+13], [QRD13], [RR13], [SH13], [ASN14], [APM+14], [BNB14], [BLL+13], [CWD+14], [GTBW14], [JBMS14], [MZM+14], [QPB+14], [WGS+14], [AGV15], [LGCR15], [MMFR+15], [MFTR+15], [RGM+15], [OGRT15], [tBLLLV15a], [tBLLLV15b], [DMSEB15], [HPHLT15], [DKL+16], [DBN16], [GAT+16], [HMGB16], [KTS16], [MBD+16], [NC16], [NN16], [SWK+16], [TC16], [tBLLLV16], [WLS+16], [QRD13], [SRD16] , [SRD17] [MOP+19], [HPF20a], [SFE+21]	119
Goal-oriented	[DS07], [DSD07], [DS08], [SMD+12] , [DMSEB15], [MFTR+15] [MBD+16]	7

Elements for Enhancing Variability

Table 2.4 presents the classification of the primary studies with respect to the usage of constructs or constraints to enhance variability descriptions. Each row in the table presents one of the categories described in Section 2.3.1, the references to the publications and the total number of publications. Accordingly, 64 publications in the collection of selected publications present variability models enhanced with constructs or constraints. From these 64 publications, 18 employ constraints to optimize an objective function, and other 17 studies include Non-functional information in the variability model via constraints. Given these data, the 46% of the selected publications include constraints to enrich variability models. Thus, these results confirm the need of a mechanism for increasing the expressiveness in variability models.

2.4.2 What types of constraint systems are used to encode variability models for analysis purposes?

This section presents the mapping and classification of publications regarding the transformation-centered facets in the framework, question **SMS.Q2**, and its derived subquestions. The following two subsections show the results retrieved consequently with the two subsections and the two facets in the transformation view.

Transformation Rules

The first result concerning the rules to encode variability models is that not all the primary sources propose their own set of transformation rules. Instead, there is a set of well-known (and cited) publications introducing transformation rules, or syntactic rules to encode variability models into constraints or formulas. Table 2.5

Table 2.4: Results of the classification regarding the SPLE constraints.

Type	Category	Publications	#
Construct	Quantified implies	[QRD13], [SRD16]	2
	Conditioned inclusion/exclusion	[DTS+14], [KOD13], [Kru07]	3
	Visibility	[EPAH09a], [FO09], [STJ09], [SMDD10], [tBLP13], [JBMS14], [WN09], [WNS09], [PRM+12], [NBE12], [MGH+11]	11
Constraints	Global	[KOD10a].	1
	Traceability	[MRM+12]	1
	Time	[BLL+13], [WZZ15], [SRD17], [ASS+19]	4
	Soft	[BDRG10], [SMD+12], [MZM+14], [MMFR+15], [MFTR+15], [DMSEB15], [MBD+16]	7
	NFRQ	[BTRC05a], [WSWN07], [WSC+07], [WSWN08], [WBS+10], [BDRG10], [RFBC10], [MHG+11], [SMD+12], [RFBRC+12], [APM+14], [WGS+14], [DMSEB15], [MFTR+15], [OGRT15], [DKL+16], [MBD+16]	17
	Optimization	[WSWN07], [WSWN08], [SGW10], [MHG+11], [GRF+12], [Hei13], [SH13], [GTBW14], [WN09], [WNS09], [PRM+12], [RFBRC+12], [EPAH08], [EPAH09b], [LDSSH15], [LDSSHC15a], [LDSSHC15b], [OGRT15]	18

recalls the list of publications of documents containing new proposals or extensions of transformation rules described in Section 2.3.2. Each row in Table 2.5 represent a publications in the list. The columns in the table contain the following data:

1. An identifier between R1 and R27 in accordance with Section 2.3.2.
2. The reference of the publication.
3. If the set of rules extends other rules, the third column contains the identifier of the extended publication.
4. The collection of variability modeling concepts that can be encoded as constraint problems applying the current set of rules.
5. The collection of studies using the set of transformation rules.

To explain the contents in Table 2.5, consider for example the rules in R5 [BTRC05a]. These rules introduce a set of rules to transform a feature model into a constraint program and perform feature model analysis. In this first proposal, authors recall that as future work, they would include rules for cross-reference constraints (includes/excludes). Later, in rules R8 [BSTRC06b], Benavides *et al.* extend their first proposal including rules for cross-reference constraints. More recently, other publications such as [TBD+08, WSWN08, WGS+14] use the rules introduced in [BSTRC06b] without further changes.

Table 2.5: Transformation rules and supported concepts.

Id	Reference	Extends	Implication	Mutex	Hierarchy one-to-one	Hierarchy one-to-many	Non-Boolean	Tree-structure	Multiplicity	Attributes	Complex empressions	Publications	#
R1	[SS02]		●	●	●	◐	o	o	o	o	o	[SS02]	1
R2	[Man02]		●	●	●	◐	o	●	o	o	o	[Man02], [TC09]	2
R3	[VK02]		●	●	●	◐	o	●	o	o	o	[VK02]	1
R4	[ZZM04]		●	●	●	◐	o	●	o	o	o	[ZZM04], [ZMZ06], [ZYZJ08], [YZZM09], [ZZM11]	5
R5	[BTRC05a]		●	●	●	◐	●	●	o	o	o	[BTRC05a], [BTRC05b]	2
R6	[Bat05]		●	●	●	●	o	●	o	o	●	[Bat05], [TBKC07], [MWC09], [MBC09], [CHH09],[STJ09], [TBK09], [BDRG10], [TBG13], [RR13], [ASN14], [LGCR15], [CK05], [MWCC08],[MPH+07], [Wan10], [ACSW12], [LHCF+13], [HPP+13], [CWD+14], [AGV15], [Seg08], [KTS16]	23
R7	[SZFW05]		●	●	●	◐	o	●	o	o	o	[SZFW05]	1
R8	[BSTRC06a]	R5	●	●	●	◐	●	●	o	●	o	[BSTRC06a], [TBRC06], [WSB+08], [TBD+08], [WDSB09], [WBS+10], [SGW10], [WGS+14], [Hei13], [BLL+13], [PLP11], [LHCF+13], [HPP+13], [CWD+14], [AGV15], [TC09] , [Seg08], [DBN16], [MBD+16], [NC16], [NN16], [OGRT15], [TC16]	23
R9	[BSTRC06b]	R9	●	●	●	●	●	●	o	●	●	[BSTRC06b], [WSWN07], [WSC+07], [WSWN08], [GTBW14]	5
R10	[CW07]		●	●	●	●	o	●	o	o	●	[CW07], [XHSC12], [PLP11], [Wan10], [ACSW12], [BDA+13], [ZKY+14], [JBMS14], [TC09], [Seg08]	10
R11	[DS07]		●	●	●	●	●	●	o	o	●	[DS07], [DSD07], [SDD+09], [SRM09], [APM+14], [BNB14]	6
R12	[DS08]	R11	●	●	●	●	●	●	o	●	●	[DS08], [SM12], [SH13]	3
R13	[EPAH08]		●	●	●	●	●	o	o	o	●	[EPAH08] [EPAH09a], [EPAH09b], [EFV+13]	4

Continue on the next page

Table 2.5: Transformation rules and supported concepts. (cont.).

Id	Reference	Extends	Implication	Mutex	Hierarchy one-to-one	Hierarchy one-to-many	Non-Boolean	Tree-structure	Multiplicity	Attributes	Complex empressions	Publications	#
R14	[FO09]		•	•	•	◐	o	o	o	o	o	[FO09]	1
R15	[WNS09]		•	•	•	•	o	•	o	o	•	[WNS09], [WN09], [WZZ15]	3
R16	[KOD10b]	R6	•	•	•	•	•	•	•	•	•	[KOD10a], [KOD10b], [KOD13]	3
R17	[RFBC10]	R9	•	•	•	•	•	•	o	o	o	[RFBC10], [RFBRC+12], [MBD+16]	3
R18	[SMDD10]		•	•	•	•	•	•	•	•	•	[SMDD10], [MLHS+11], [SMD10], [MSDL11], [SMD+11b], [MSD12a], [MSD+12b], [RGM+15], [MSD11]	9
R19	[CBH11]	R6	•	•	•	•	•	•	o	•	•	[CBH11]	1
R20	[DFH11]	R6	•	•	•	•	•	•	•	o	•	[DFH11], [NBE12] [BDA+13], [ZKY+14]	4
R22	[MGH+11]		•	•	•	•	•	•	o	o	•	[MGH+11], [MDSD14], [MBD+16], [QRD13], [QRD13], [DTS+14], [ASS+19]	7
R23	[SMD+12]		•	•	•	•	•	•	o	o	•	[SMD+12], [MFTR+15], [DMSEB15], [MMFR+15]	4
R24	[SHTB07]		•	•	•	•	•	•	o	o	•	[SHTB07] [PSK+10], [HBG11], [MRM+12], [tBLP13], [MPH+07], [JBMS14], [tBLLLV15a], [tBLLLV15b], [HMGB16], [tBLLLV16]	11
R25	[BSTRC07]	R9	•	•	•	•	•	•	•	•	•	[BSTRC07], [FBGR13], [BTCS13], [QRD13], [QPB+14]	5
R26	[TMV+16]		•	•	•	•	•	•	o	o	•	[TMV+16]	1
R26	[WLS+16]		•	•	•	•	•	•	•	o	•	[WLS+16], [SWK+16]	2
R27	[WLS+16]		•	•	•	•	•	•	o	•	•	[WLS+16], [SWK+16]	2
R28	[MOP+19]	R6	•	•	•	•	•	•	o	•	o	[MOP+19]	1

Constraint Systems

Figure 2.2 summarizes the classifications regarding the constraint systems used in the reviewed publications. The results show that 84 publications in the collection of selected papers use Boolean variables and Boolean constraints to represent variability models. Then, the constraint system over boolean is the most frequently used. Constraint systems with integer variables and constraints are the second most frequent constraint systems in the

review with 72 publications. At the other end of the spectrum, the least frequently used constraint systems are symbolic domains (1 publication), and constraint systems with special variables (2 publications). Additionally, a small number of publications (8 publications) use constraint systems with Real numbers. Eight studies did not report the information about the implementation of the corresponding approaches. To conclude, it should be noted that the results in Figure 2.2 show a preference for Boolean variables and Boolean constraints.

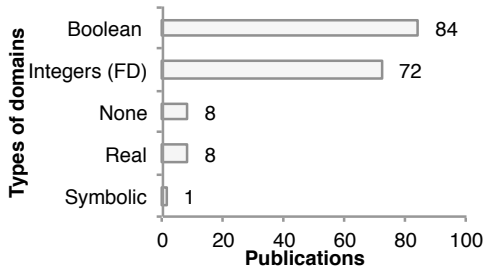


Figure 2.2: Distribution of the usage of solver-provided domains.

2.4.3 What are the characteristics of the solvers supporting variability management?

This section presents the mapping and classification of publications regarding the support-centered facets in the framework, question **SMS.Q3**, and its derived subquestions. Consequently, the two subsections below presents the classification regarding the two facets centered on solvers.

Figure 2.3 displays the results regarding the number of solvers reported by the primary studies. The majority of the reviewed studies uses one single solver (59%) and the 26% of the publications do not report any solver at all. Therefore, this 26% falls in the category *None*. The remaining 15% of publications report the usage of more than one solver distributed as follows: 8% studies use two solvers, 4% use three solvers, and 3% use more than three solvers.

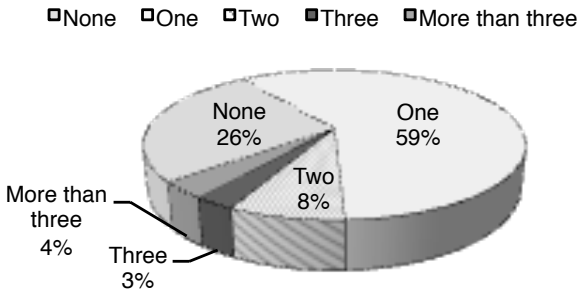


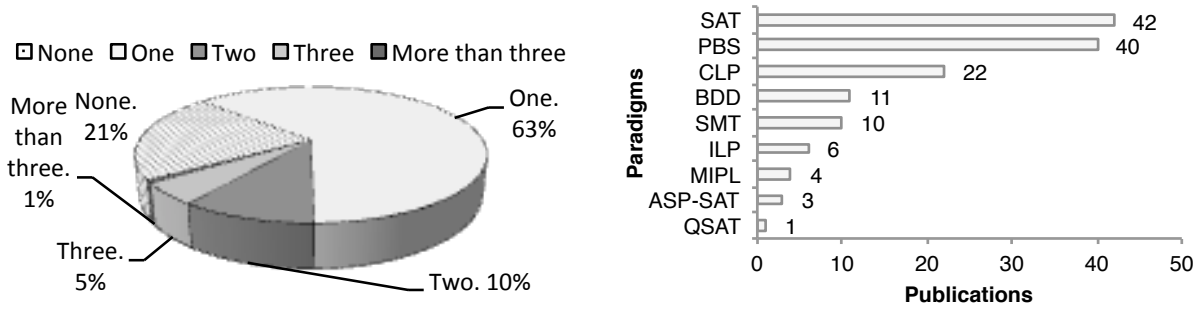
Figure 2.3: Distribution of the number of solvers reported by the primary studies.

Solving paradigm

Figure 2.4 summarizes the results of the classification regarding the solving paradigm. This figure has two subfigures. Figure 2.4a shows the number of solving paradigms reported by the primary studies. Figure 2.4b depicts the most used solving paradigms to encode variability models to perform analysis tasks.

As shown in Figure 2.4a, most publications report the usage of a single solving paradigm for encoding variability models (63%). However, these results also show the usage of more than one solving paradigm in the 16% of the papers. The distribution of the number of solving paradigms for encoding variability models is the following: 10% of the approaches employ exactly two paradigms; the most common combination of paradigms being Constraint Satisfaction Problems (CSP) and integer linear programming solvers (ILP). Additionally, 5% of the primary studies use three solving paradigms with CSP, Binary Decision Diagrams (BDD), and Satisfiability Problems (SAT) as the most frequent combination of paradigms. Particularly, only one publication [PLP11] reported the usage of more than three paradigms as the authors report the comparison of approaches for automated analysis of feature models. The remaining 21% of the publications report the usage of solvers, but not provide plus information about solver names, nor paradigms.

Figure 2.4b shows the frequency of each solving paradigm. In accordance with the evidence, variability models are most frequently encoded as satiasfibility problems (SAT, 42 studies), constraint satisfaction problems (CSP, 40 studies), and Constraint logic Programming Problems (CLP, 22 studies). On the other hand, the least common paradigms are the two special cases of SAT problems: Q-SAT (quantified SAT problems) and ASP-SAT (answer set satisfiability problems) with one and three studies, respectively.



(a) Distribution of the number of solving paradigms reported by primary studies.

(b) Most used solving paradigms for variability management.

Figure 2.4: Results of the classification regarding the solving paradigm.

List of solvers

This literature review gathered evidence for the usage of 36 solvers supporting variability management tasks. Table 2.6 and Figure 2.5 present the results. On the one hand, Table 2.6 list the information of those 36 solvers. Each row in the table presents the name of the solver, the solving paradigm, approach used for implementing the solver and an URL to find more information.

Figure 2.5 presents the list of solvers together with the number of publications citing them. The figure depicts a bar graph instead of a pie chart, because some proposals use more than one solver. Accordingly, the most used solvers are Sat4j (21 studies), CHOCO (18 studies), and GNU Prolog (13 studies). At the other end of the spectrum, there are other 25 solvers used by one, two or three works. One particular case is the Prolog language because in the reviewed publications there are at least three different implementations of the language: GNU Prolog, SWI Prolog, and SICStus Prolog. Other studies just mention the Prolog language without referencing

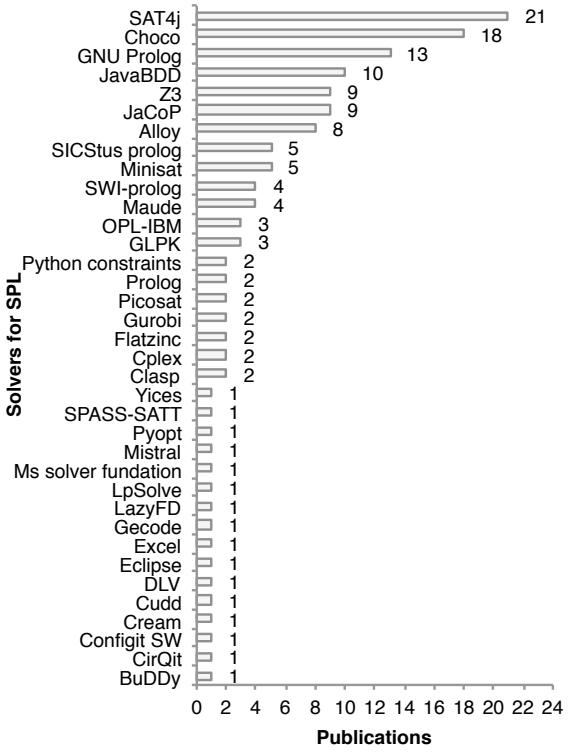


Figure 2.5: Solvers used to support variability management and the number of publications citing them.

a particular implementation. Considering these three different implementations as one single language, it is possible to conclude that Prolog based solvers are the most used solvers form variability management.

Table 2.6: Solvers used to support variability management.

Solver	Paradigm	Implementation	URL
Alloy-analyzer	SAT	Language	http://alloy.mit.edu/alloy/
Buddy	BDD	Library	http://buddy.sourceforge.net/manual/main.html
Choco	PBS	Library	http://choco-solver.org/
CirQit	QSAT	Tool	http://www.cs.utoronto.ca/ alexia/cirqit/
Clasp	ASP, SAT	Library, tool	http://www.cs.uni-potsdam.de/clasp/
Configit SW	DBB	Tool	http://configit.com/about/history/
CPLEX Optimizer	ILP, MILP	Tool	http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html
Cream	PBS	Library	http://bach.istc.kobe-u.ac.jp/cream/
Cudd	BDD	Tool	http://vlsi.colorado.edu/ fabio/

Continue on the next page

Table 2.6: Solvers used to support variability management. (cont.).

Solver	Paradigm	Implementation	URL
DLV	CLP	Language	http://www.dlvsystem.com/
ECLiPSe	CLP	Language	https://sourceforge.net/projects/eclipse-clp/
Excel	ILP	Tool	https://support.office.com/en-us/excel
MinizincFlatZinc	Multi	Language	http://www.minizinc.org/
Gecode	PBS	Library	http://www.gecode.org/
GLPK	MILP	Library	https://www.gnu.org/software/glpk/
GNU Prolog	CLP	Language	http://www.gprolog.org/
Gurobi	ILP, MILP	Library	http://www.gurobi.com/index
JaCoP	PBS	Library	http://jacop.osolpro.com/
JavaBDD	BDD	Library	http://javabdd.sourceforge.net/
LazyFD	SAT	−−	−−
LpSolve	MILP	Library	http://lpsolve.sourceforge.net/5.5/
Maude	SAT	Language	http://maude.cs.illinois.edu/w/images/0/0d/Maude-book.pdf
Minisat	SAT	Tool	http://minisat.se/
Mistral	PBS	Library	http://homepages.laas.fr/ehebrard/mistral.html
MSDN	PBS, LP	Library	https://msdn.microsoft.com/en-us/library/ff524509(v=vs.93).aspx
OPL-IBM	PBS, LP	Language	http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud
Picosat	SAT	Library	https://pypi.python.org/pypi/pycosat
Prolog	CLP	Language	−−
Pyopt	COP	Library	http://www.pyopt.org/
Python constraints	PBS	Library	https://labix.org/python-constraint
SAT4j	SAT	Library	http://www.sat4j.org/
SICStus prolog	CLP	Language	https://sicstus.sics.se/
SPASS-SATT	SAT	Tool	http://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench/classic-spass-theorem-prover/tutorial/
Swi Prolog	CLP	Language	http://www.swi-prolog.org/
Yices	SMT	Library, tool	http://yices.csl.sri.com/
Z3	SMT	Tool	https://github.com/Z3Prover/z3

2.4.4 What are the characteristics of the variability-management tools?

The first result regarding the usage of software tools is that not all the reviewed publications provide evidence of the implementation of their proposals. Nevertheless, most of them, the 80% of the selected publications present evidence of the usage of constraint-based software by (1) explicitly presenting a software element as a contribution of the publication; or (2) including a *results section* containing the results obtained from experiments related to the implementation and execution of the contribution of the publication. The results presented in Tables 2.7 summarizes the characteristics of the most relevant software tools in the domain.

Software Tools

Table 2.7 contains the characteristics of 15 software tools that report the usage of constraint programming to perform variability management tasks. Each row in the table contains a software tool. The columns in the table contain the following data: An identifier between R1 and R27 in accordance with Section 2.3.2.

1. The name of the tool and the year of the publication.
2. The modeling paradigms supported by the tool.
3. The type of implementation in accordance with the categories in the support facet described in Section 2.3.3.
4. The functionalities related with variability management provided by the tool. Section 2.3.3 describes the categories in this column.
5. Four features the tool may have: Graphical User's Interface (GUI), open source code, user's documentation, and examples. These columns contain a • if the tool has the feature and − in the opposite case.
6. The URL of the tool.

The frequency of feature-oriented models as supported paradigm in Table 2.7 confirms that *feature models* are the preferred paradigm for modeling variability. However, other paradigms, such as, decision-oriented models and variation point-oriented models are also present.

Table 2.7: Constraint-based variability management tools.

Name	Year	Paradigm	Imp.	Functionality	GUI	Open source	User doc.	Examples	URL
AHEAD	2004	Feature models	Stand-alone	Modeling, Configuration, Derivation	•	•	•	•	http://www.cs.utexas.edu/users/schwartz/ATS.html
CardyGAn	2016	Feature models	Plug-in, Eclipse	Modeling, Analysis, Configuration	•	•	•	•	https://github.com/Echtzeitsysteme/cardygan
ClaferMOO	2012	Feature models	Stand-alone	Non-Functional Properties	•	•	•	•	https://github.com/gsdlab/claferMooStandalone

Continue on the next page

Table 2.7: Constraint-based variability management tools (cont.).

Name	Year	Paradigm	Imp.	Function	GUI	Open source	User doc.	Examples	URL
DecisionKing	2007	Decision models	Plug-in		•	–	–	–	–
FaMa	2007	Feature models	Variability management tool	Modeling, Verification, Configuration	•	•	•	•	http://www.isa.us.es/fama/?FaMa_Framework
FeatureIDE	2005	Feature models	Plugin, Eclipse	Modeling, Configuration, Verification	•	•	•	•	http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/
Fuji	2011	Feature models	Standalone	Derivation, verification	–	–	•	•	http://fosd.net/fuji
Gears	2007	Variation point models	Variability management tool	Modeling, Configuration, Verification	•	–	•	•	https://biglever.com/solution/gears/
HOLMES	2001	UML	Prototype	Modeling, configuration	•	–	–	–	–
Kesit	2008	Feature models	Standalone	Testing	–	–	–	–	–
Pacogen	2011	Feature models	Standalone	Testing	!	•	–	–	http://people.rennes.inria.fr/Arnaud.Gotlieb/resources/Pacogen/Pacogen.html
ProVelines	2013	Variation point models	Standalone	Verification	•	•	•	•	https://projects.info.unamur.be/fts/provelines/
Pure::variants	2006	Variation point models	Variability management tool	Modeling, Verification, Configuration, Derivation	•	–	•	•	http://www.pure-systems.com/pure_variants.49.0.html
SPLOT	2009	Feature models	Variability management tool	Modeling, Verification, Configuration.	•	•	•	•	http://www.splot-research.org/
VariaMos	2012	Multi paradigm	Variability management tool	Modeling, Verification, Configuration, simulation.	•	•	•	•	http://variamos.com/home/
VMWare	2007	Feature models	Prototype	Model Checking, verification	–	–	–	–	–

2.5 Lessons Learned

The objective of this systematic mapping study was to provide an overview of the research of the constraint-oriented approaches used in variability management with a particular emphasis on the information and characteristics of the variability represented as constraint problems. Section 2.4 presented the results of the classification. This section answers the research questions, highlights the most relevant results, and discuss the conclusions obtained from this study.

2.5.1 Variability Modeling

This review considered two complementary approaches to determine the variability concepts: the modeling paradigm, and the variability concepts in the modeling language.

First, the conduction of this systematic mapping study identified similarities among the ground characteristics of variability languages. These similarities include the characteristics of the variability units, the set of variability relationships, the structure of the model, the configuration semantics, etc. Then, the inclusion of the concept of modeling paradigm seemed natural to delineate the similarities among variability languages. Following, this thesis defined the modeling paradigm as the approach employed to capture variability in the form of models, programs or code. From that point, similar languages were mapped to modeling paradigms.

Second, when observed under the lens of modeling paradigms, variability modeling languages showed differences in terms of syntactic elements and syntactic rules. However, the languages in different modeling paradigms have similarities regarding their semantics. The similarities among the semantics are evident when no matter the paradigm, all reviewed modeling languages are encoded using a constraint-based formalism. Then, this mapping study starts to consider the term *variability concept* to name different syntactic elements with similar or equivalent semantics. As result, the classification framework includes a glossary describing most of the concepts used to represent variability (*cf.* Section 2.3.1).

2.5.2 Transforming Variability Models into Constraint Programs

Transforming variability models into constraint programs is the first step to automate the analysis tasks. The mapping and analysis of the selected publications showed that each reviewed publication can either propose, extend, or use a set of transformation rules. The reader should note that the majority of reviewed documents use the transformation rules proposed by other authors. Then, this work concluded that there is a a well-known assortment of transformation rules and proceeded to characterize them.

Table 2.2 lists the set of publication that propose or extend a set of transformation rules. This table was included in the transformation framework to classify each publication regarding the rules used in the development of a constraint-based approach. Then, the results in Section 2.4.2 presents the Table 2.5 where the transformation rules are mapped with the supported variability concepts and the publications using those rules. In this sense, Table 2.5 provides information about the constraints that could be obtained by using a particular set of transformation rules.

The obtained results reinforce the idea that constraints are a flexible, expressive and suitable approach to encode variability for analysis purposes, capturing the information on current variability models and allowing the inclusion of more problem-related information. Nevertheless, the apparition of many different rules for the same purpose suggests that there is still a gap in the construction of a unified transformation approach able to gather the advantages of the different proposals, as well as useful for including new variability information shaped as constraints.

2.5.3 Solvers Supporting Variability Management

The results regarding the analysis and classification regarding solvers showed that variability management is supported by different specialized solvers, solving paradigms, and constraint systems. The results in Section 2.4.3 showed that the usage of solvers is diverse. Table 2.6 lists 36 different solvers used to support variability management. There is no preference in the implementation of these solvers, because the implementation of these 36 solvers ranges between solvers implemented as libraries, tools, or provided by programming languages.

The transformation rules reported in Table 2.5 produced constraint problems in Boolean, Integer, Real and Symbolic constraint systems. Meanwhile, the solving paradigm in the reviewed papers ranges over a variety of paradigms such as ILP, MILP, SAT, SMT, BDD, CSP, and CLP. In the case of the support for different constraint systems, the results showed that most of the publications use the variables and constraints provided by the solver. Then, there is no implementation of specialized constraint systems for supporting variability management. To the best of our knowledge, there is no one solving parading or solver specially designed for supporting variability management. Nevertheless, considering that each different paradigm and solver has its strengths, it would be useful to drive the research efforts to combine paradigms for exploiting the advantages and strengths of distinct paradigms and solvers for the problems and situations inherent to variability management.

2.5.4 Software tools supporting constraints in SPLE

Another result of this mapping study is the evidence of the support of variability management in different types of tools. Moreover, most of the publications report the development or usage of a software tool aiming to a variety of applications. However, there is no tool supporting variability management in an holistic way. Consequently, to summarize the results regarding tools and to provide a global view that serves as a start point for practitioners in the area, this section presents the Table 2.8. This table synthesizes the results considering the functionality of the tools and the following information:

- A list of the solvers supporting each type of functionality. The solvers in this list come from Table 2.6.
- A list of tools. This list also contains a reference to the publications classified in the topmost evaluation level category for each software tool. Categories for the evaluation level are described in the scale proposed by Petersen *et al.* [PVK15] and described as follows:
  - *None.* No empirical evidence, the evidence may be provided from observations, demonstration or arguments.
  - *Toy-example.* The evidence is obtained by demonstration on toy-examples.
  - *Academic.* The evidence is obtained by working out on case studies employed on other publications or repositories.

  - *Academic (Big-set).* Evidence obtained using case studies reported as real life size, or big. These cases are randomly generated or obtained from public repositories.
  - *Industrial.* Evidence obtained using case studies inspired from industrial partners.
  - *Industrial practice.* The contributions of the publication is used in an industrial context.
- The evaluation level of publications including the references to publications.
- Additional information or special remarks.

Table 2.8: A global overview of tools supporting variability management.

Application	Variability system	Solvers	Software tools	Evaluation Level	Additional information
Analisys	SPL, DSPL, PL.	Alloy-analyzer, Buddy, Choco, CirQit, Clasp, Configit SW, Cream, Cudd, GLPK, GNU Prolog, JaCoP, JavaDBB, Minisat, Mistral, OPL-IBM, Picosat, Prolog, SAT4j, SICStus prolog, Swi Prolog, Z3.	FaMa (industrial) [RFBRC+12], FeatureIDE (academic) [TBK09], Pure::variants (academic) [Wan10], TVL (DSL) (industrial) [CBH11, BDA+13], S.P.L.O.T (academic big-set) [MBC09], VariaMos (academic) [MMFR+15]	Industrial [RFBRC+12, MZM+14, LGCR15, CBH11, BDA+13],	Variability Analysis is often executed together with configuration, derivation, specification and simulation. Analysis is one of the most frequent functionalities reported in the reviewed publications.
Configuration	PL, SPL, DSPL, CS.	Alloy-analyzer, Choco, Clasp, Configit SW, DLV, ECLiPSe , FlatZinc, Gecode, GNU Prolog, JaCoP, JavaBDD, LazyFD, Minisat, SAT4j, SICStus Prolog, Z3.	AHEAD (academic) [Bat05], ClaferMOO (academic) [ZKY+14], FaMa (academic) [WGS+14], Familiar (academic) [ACLF11], Pure::variants (academic) [Wan10], S.P.L.O.T. (academic) [MBC09], VariaMos (industrial-practice) [MDSD14]	Industrial-practice [WSB+08, MDSD14], Industrial[BNB14, STJ09, XHSC12, BLL+13, SMDD10, SMD+11b].	Configuration is often reported together with analysis, specification, simulation, verification. Configuration is the second most frequent functionality reported by the selected publications.
Derivation	SPL, DSPL, PL.	Choco, Excel, GNU Prolog, Picosat	Fuji repository (academic) [ASN14]	Industrial-practice (anonymous tool) [DSD07], Academic [ASN14, SGW10]	Derivation is often implements together analysis, also is one of the less frequent activities.

Continue on the next page

Table 2.8: A global overview of constraint-based approaches for variability management. (cont.).

Application	Variability system	Solvers	Software tools	Evaluation Level	Additional information
Variability Modeling	SPL, DSPL, PL, CS.	Alloy-analyzer, Choco, CirQit, Clasp, GNU Prolog, JavaBDD, LPSolve, Maude, Minisat, Swi Prolog, SAT4j, Z3.	VariaMos (industrial) [MSDL11], academic [MMFR+15], S.P.L.O.T. (academic big-set) [MBC09], Familiar (academic) [ACLF11], FeatureIDE (academic) [TBK09], Clafer (industrial) [CBH11], TVL (academic) [BDA+13], FLAN (academic) [tBLP13].	Industrial-practice [WSB+08], Industrial [MSDL11, CBH11, BLL+13, SMDD10, SMD+11b]	Variability Modeling is a functionality often implemented together with analysis, configuration, and simulation. Most of the support is provided by domain specific languages (Familiar, Clafer, TVL, FLAN). The specification is also supported by formalization such as formal semantics for feature models [MCHB11] and configuration problems [CHH09].
Verification	SPL, PL	Alloy-analyzer, Choco, GNU Prolog, Minisat, Maude, Prolog, SAT4j, Swi Prolog, Z3.	AHEAD (academic) [TBKC07], Decision King (academic) [TBG13], FLAN (academic) [tBLP13], VariaMos (industrial) [MLHS+11, SMD+11b].	Industrial, [MLHS+11, SMDD10, SMD+11b], Academic big-set [SM12, MSD+12b].	This activity is often implemented together with specification and configuration.
Testing	SPL, PL	Alloy-analyzer, Choco, Minisat, SAT4j, SPASS-SAT, Yices, Z3	AHEAD (academic) [UGKB08], Kesit (academic) [UGKB08, UKB10], FeatureIDE (academic) [AGV15], Pacogen (industrial) [MGSH13].	Industrial [MGSH13, PSK+10], Academic big-set [HPP+13, GTBW14].	Most publications aim to solve testing related problems without including other activities.
Simulation	DSPL	GNU Prolog, Swi Prolog	VariaMos (academic big-set) [MFTR+15, MMFR+15]	Academic big-set [MFTR+15], acaedmic [MMFR+15]	Proposals aiming to solve simulation related problems are scarce, and both of them deal with the simulation of DSPLs
Synthesis	SPL	JavaBDD, SAT4j	Anonymous tool (academic big-set)[ACSW12]	Academic big-set [ACSW12].	Synthesize a constraint program is to obtain a feature model from a formal specification. Two documents [CW07, ACSW12] contain proposals to perform synthesis.

2.6 Concluding Remarks

This chapter presented the results of a systematic mapping study on the application of constraint-based approaches in the life-cycle of variability intensive systems.

The process to conduct the systematic literature review followed the guidelines and recommendations from Petersen *et al.* [PVK15] and Kitchenham *et al.* [KPP+02]. As result, the search and filtering stages identified 137 papers published between 2000 and 2020.

Section 2.3 presents the classification framework used for analyzing and extracting data. The classification framework is structured in a hierarchical fashion with three views and twelve facets. The hierarchical structure of the classification framework guided the reviewing and reporting process. Table 2.1 summarizes the classification framework.

The definition of the classification framework was an iterative process resulting in a framework that synthesizes the knowledge gathered from dozens of papers regarding variability management using any constraint-based approach. Notably, this framework presents a glossary of variability modeling concepts. These modeling concepts serve as a guide of what are the modeling requirements in the community, considering what pieces of research are already using in their models. The glossary also includes concepts, such as the quantified implies, that may be considered as particularities of modeling languages. However, this framework includes these concepts as they could represent as well new trends in the design of variability modeling languages.

The collection of concepts in Section 2.3.1 contributes to describe variability-intensive systems more accurately. Nevertheless, to the best of our knowledge, there are not initiatives aiming to integrate all these concepts in an unified language. One of the possible reasons is the trade-off between expressiveness and the analysis capabilities of the language. However, an unified variability modeling language may improve the expressiveness and accuracy of variability specifications.

The literature review evidenced the diversity in the approaches to encode variability models as constraint satisfaction problems. The results showed that there are at least 23 different publications containing a set of transformation rules or an extension to a set of rules. One challenge in the research area is to consider this diversity and develop an standard analysis library. The development of a standard analysis library shall propel the usage of variability management tools and unite the variability community efforts to provide a robust common analysis core.

There are two types of tools supporting constraint-based approaches: solvers and other software tools. Despite there is no one solver specially designed for variability management, there is broad support for solving constraint problems. Also, there is no a particular solving paradigm for variability management. On the contrary, this mapping study showed that solvers in variability management range in a variety of paradigms, constraint systems and implementations. One challenge regarding solvers is to address the difficulty of exploiting the strengths of multiple solvers while solving a single complex problem.

Part II

Studies and Results

Chapter 3

From the Evaluation of the HLCL Framework Towards Coffee

This chapter shares content with the paper *On the Ontological Expressiveness of the High-Level Constraint Language for Product Line Specification* [VMS18]

This chapter serves of three goals. First, it presents the results of evaluating the HLCL framework emphasizing on the theoretical evaluation of the High-Level Constraint Language (HLCL), the core of the HLCL framework. This theoretical evaluation measures the language's expressiveness as a variability modeling language. Second, the reader will find the discussion concerning the evaluation's results, and the drawbacks addressed by the contributions in this thesis. Finally, the chapter closes presenting the conceptual model of Coffee and an overview of each layer within the framework.

The following sections present (1) an overview of the theory grounding the evaluation, *i.e.,* the theory of ontological expressiveness; (2) the design and execution of the theoretical evaluation following a GQM approach; (3) the analysis and discussion of the evaluation, and (4) an overlook of the Coffee Framework.

3.1 Motivation and Challenges

The first phase in the development of this thesis included the evaluation of the state-of-the-art HLCL framework. This framework takes its name from the High-Level Constraint Language (HLCL), an abstract language proposed to represent variability models from different notations developed at the *Centre de Recherche en Informatique* (CRI). The HLCL framework evolved through the research of Djebbi & Salinesi, Salinesi *et al.*, Mazo *et al.*, and Munoz-Fernandez *et al.* [DS08, SMD+11a, MSD11, MFTR+15]. This framework is the core of early implementations of the VariaMos tool-suite [MMFR+15], also developed by the CRI.

The idea of including one extra-step to represent variability models using an abstract constraint language developed in the HLCL framework is consistent to the hypothesis of the applicability of intermediate representations developed in this thesis. Hence, the research presented in this thesis started with an evaluation of the HLCL framework.

The HLCL framework had proven useful for representing different variability languages [SMD+11a, MSD+12b, MFTR+15], and for supporting analysis tasks such as verification and configuration of variability models in those languages [MSD+12b, SM12, MFTR+15]. However, the HLCL framework and its implementation in the

VariaMos tool suite are part of the ecosystem of non-compatible variability management tools. Then, the evaluation of the extent of the HLCL framework is primordial for better understanding its drawbacks and explore the solution perspectives.

The evaluation conducted covered two complementary perspectives: the engineering of a variability management tool, and the evaluation of the HLCL's expressiveness. The first perspective consisted on the participation of the author of this thesis as a member of the team engineering the VariaMos Tool suite at the CRI. The main goal of this participation was the design and implementation of a graphical representation of the high-level constraint language as modeling language called the *Constraint Graphs*. Additionally, the exercise of engineering a variability management tool included the development and evaluation of a method to identify and fix inconsistencies on variability models represented as constraint graphs. The proposal and its evaluation are part of an upcoming publication [MVSD].

The second perspective consisted on the evaluation of the HLCL as modeling language for variability-intensive systems. HLCL was previously evaluated regarding it's capability to encode other variability languages, i.e., feature-oriented languages [SMD+11a, MSD+12b]; variation point oriented languages; decision-oriented languages [MSD+12b]; and goal-oriented languages [SMD+12, MFTR+15]. However, the question: *Is HLCL expressive enough to encompass any product line model?* is still unanswered. Considering that the previous evaluations of HLCL followed a practical point of view, and the universality of the research question, we opted to provide an evaluation from a theoretical point of view. Then, this theoretical evaluation required the definition of an evaluation framework capable to answer questions, such as, *Which are the characteristics of an expressive variability modeling language?* and *How to measure the expressiveness in a variability modeling language?*. To tackle this challenge, this thesis composed an evaluation framework grounded on the theory of ontological expressiveness [WW93] and its extension to variability modeling languages [AGWH12].

First, the theory of ontological expressiveness is based on the observation that models of information systems are essentially models of real-world systems and has already been used to evaluate the expressiveness of conceptual modeling languages such as the entity-relation model [SMN+10]; UML [BJM08], the $i^\star$ language [GFG12]; and BPMN [RRIG09].

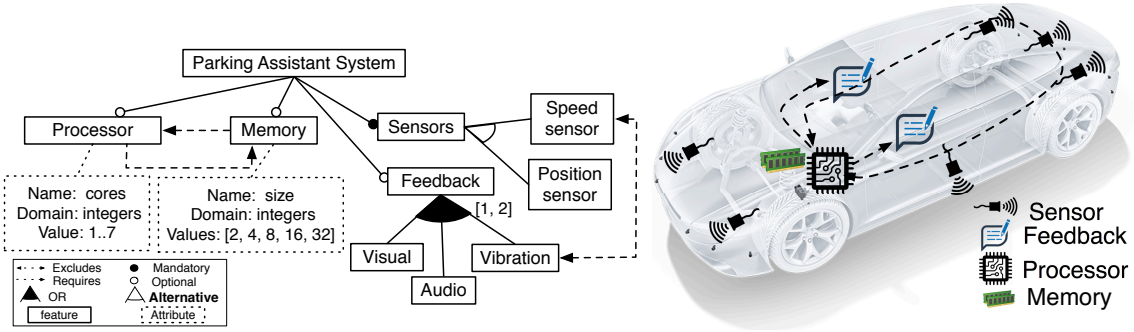
Second, the application of the theory of ontological expressiveness requires a mapping between a reference ontology and the constructs of the language under evaluation. This study uses Asadi *et al.*'s ontology as reference to perform the mapping. This ontology divides concepts between structural concepts and variability patterns. At this point, the challenge was to answer questions such as, *which constraints can be used to map structural concepts and variability patterns?* and *What is the semantics of such constraints?*

The results in the evaluation demonstrated many problematic situations with the HLCL framework. The ontological evaluation showed the HLCL weakness as variability modeling language and the practical evaluation showed that the transformation framework supporting analysis of HLCL variability representations did not solve the interoperability issues. In consequence, the final part of this chapter shows the contribution of this dissertation to solve the questions about *How intermediate representations can be integrated in a framework to ease the interoperability of variability management tools?*

3.1.1 Running Example

To illustrate the steps in the ontological study, the following paragraphs present an example of a hypothetical case of a car's Parking Assistant System (PAS). This example is a simplified extract of the feature model in [SMDD10, SMD+11a, MSDL11]. The example consists on the description of the case and a feature model to illustrate the example in Figure 4.1c. Additionally, Table 3.1 shows the specification of the PAS using HLCL and some examples of the products as solutions of the constraint program.

A car's Parking Assistant System (PAS) assists drivers to park, helping them to detect obstacles, control the speed and correct the trajectory. The PAS is composed of a processor, an internal memory slot, some sensors, and optionally some feedback devices. Sensors are used to measure the speed and position of a car through speed sensors and position sensors. A PAS may contain speed sensors and position sensors, this simplified example will not constraint the number of sensors. Feedback can be visual, auditory or vibratory, and a single product can have at most two kinds of feedback. When a speed sensor is included in a configuration, then vibratory feedback must be excluded, and vice versa. To compute the location of a car, the PAS uses the processor that can have one to seven cores. The size of the internal memory can have one of the values in the set $\{2GB, 4GB, 8GB, 16GB, 32GB\}$. Additionally, the size of the memory depends on the number of cores in a processor, the pair $\langle$`cores; memory size`$\rangle$ can have the following values $\langle$`0;0`$\rangle$, $\langle$`1;2`$\rangle$, $\langle$`2;4`$\rangle$, $\langle$`3;8`$\rangle$, $\langle$`4;16`$\rangle$, $\langle$`5;32`$\rangle$.



(a) **(b)**

Table 3.1 contains the variables and constraints used to represent the PAS product line using HLCL with the extra constraint $C_0 : PAS = 1$ encoding the root feature. The resulting constraint satisfaction problem has 396 solutions or representations of valid products. Each solution is a set of pairs *variable*, *value* (x_i, v_i). In a solution, a pair (x_i, v_i) where v_i is equal to one represents the inclusion of the component associated with variable x_i in the product. The PAS example includes non-Boolean variables for those cases when additional information related to components should be considered, as in the case of the cores of a processor and the size of the memory. Thus, this additional information is represented using integer variables with domains ranging in the set of values presented in the description of the PAS. The domains of `Cores` and `Size` contain a zero to represent those products with no processor, nor memory. For instance, a valid product can be represented by the set $P =$ `{Processor, Cores=1, Memory, Size=2, Sensors, PositionSensors}`. This example does not show the variables excluded from the product because they are assigned to zero in the corresponding solution (more examples in Table 3.7).

Table 3.1: HLCL specification for the Parking Assistant System

Variables and domains
`[PAS, Sensors, SpeedSensors, PositionSensors, Processor, Feedback, Visual, Audio, Vibration, Memory]` $\in \{0,1\} \wedge$ `Size` $\in \{0,2,4,8,16,32\} \wedge$ `Cores` $\in \{0..7\}$
Constraints
C_0 : `PAS` $=$ `1` , C_1 : `PAS` $=$ `Sensors` , C_2 : `PAS` $\geq$ `Processor`, C_3 : `PAS` $\geq$ `Memory` , C_4 : `PAS` $\geq$ `Feedback`,
C_5 : `(Memory` > 0 `)` $\Leftrightarrow$ `(Size` > 0 `)`, C_6 : `(Processor` > 0 `)` $\Leftrightarrow$ `(Cores` > 0 `)`,
C_7 : `Feedback` $\leq$ `Visual + Audio+ Vibration` $\leq 2*$ `Feedback`,
C_8 : `Sensors` $\leq$ `SpeedSensors + PositionSensors` $\leq$ `Sensors`, C_9 : `Vibration` $+$ `SpeedSensors` ≤ 1
C_{10} : `Memory` $=$ `Processor`
C_{11} : `Relation (Cores, Size) [(0,0), (1,2), (2,4), (3,8), (4,16), (5,32)]`,

3.2 Ontological Expressiveness Theory

The ontological expressiveness theory is a framework to analyze the expressiveness of conceptual modeling languages [WW93]. This theory is based on the observation that conceptual models of information systems are, essentially, models of real-world systems. The theory of ontological expressiveness has served as foundation to evaluate the expressiveness of conceptual modeling languages such as the entity-relation model [SMN+10]; UML [BJM08], the $i^\star$ language [GFG12]; and BPMN [RRIG09]. The evaluation of a conceptual modeling language requires a mapping of the language's constructs with respect to a foundational ontology [WW93, Gui13]. In conceptual modeling, a foundational ontology is understood as a formally and philosophically well-founded model of categories that can be used to articulate conceptualizations in specific engineering models [GFG12]. The mapping between ontological constructs and language constructs should focus on two sets: the set of ontological constructs and the set of language constructs, as presented in Fig. 3.1 This mapping considers two steps: the representation mapping and the interpretation mapping [WW93]. On the one hand, representation mapping serves to determine whether and how an ontological construct is represented using the language constructs. On the other hand, interpretation mapping describes whether and how a language construct stands for a real-world construct. To conclude about the expressiveness of the examined language, it is decisive to determine the presence or absence of any of the four observable defects in conceptual modeling languages: construct deficit, construct excess, construct redundancy and construct overload.

Fig. 3.1 shows the defects in conceptual modeling languages. If a language has construct deficit, then the language is ontologically incomplete. The ontological clarity of the language is undermined if a language has either construct excess, construct redundancy, or construct overload. The presence or absence of these defects can be measured with the metrics of potential ontological deficiencies proposed by Recker *et al.* [RRIG09] and described in Section 3.4.

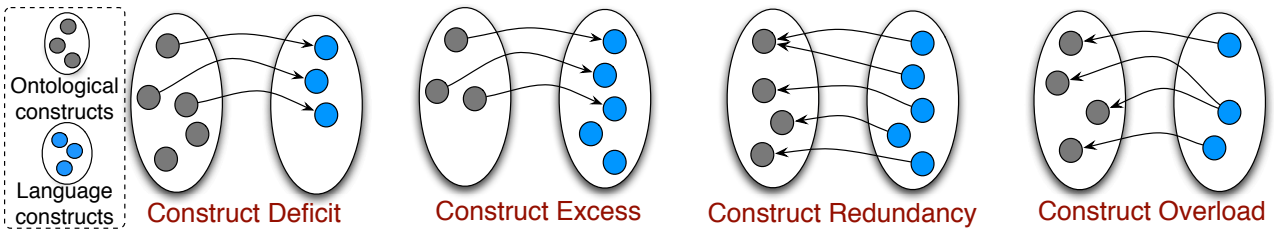


Figure 3.1: Defects in a conceptual modeling languages, taken from [WW93].

3.3 A Foundational Ontology for Variability

Three criteria were taken into account to select an ontology for evaluating the HLCL. The first criterion is the type of the language to be analyzed. As suggested by Guizzardi *et al.* in [Gui13], the type of the language (i.e., domain-independent, domain-specific) is the most important criterion for selecting the reference model in an ontological analysis. Domain-independent languages are compared to domain-independent foundational ontologies such as BWW [WW93] or UFO [GFG08]. Instead, domain-specific languages should be compared using domain-specific ontologies. HLCL has a duality with respect to this first criterion given that variability languages should be domain-independent but also have particular characteristics regarding variability. Therefore, the selected ontology should include domain-independent constructs along with variability related constructs. Thus, the second criterion is the ability to represent different variability relations. To the best of our knowledge, there exist two ontologies with concepts from a domain-independent ontology (BWW) that also include variability concepts: Reinhartz-Berger *et al.* [RBSW11] and Asadi *et al.* [AGWH12]. However, in Reinhartz-Berger *et al.*'s variability is oriented to the behavior of the system, and the variability concepts in Asadi *et al.* are of general purpose. The third criterion is the use of the ontology in similar studies. To the best of our knowledge, the ontology of Asadi *et al.* is the only one used to evaluate ontological expressiveness in variability languages (i.e., FODA and OVM). For the aforementioned reasons, the ontology proposed by Asadi *et al.* is selected for the ontological analysis of the HLCL.

Asadi *et al.*'s ontology groups concepts into variability sources and variability patterns. A *variability source* is an element in which variability may happen. The concepts considered as variability sources correspond to things, properties, lawful state space, lawful event space, and history in the BWW ontology [WW93]. Table 3.2 presents the concepts and their definitions. This table includes the definition of concepts *state*, *state law*, and *event* because they are relevant for the understanding and mapping of the *lawful state space* and *lawful event space*. Also, the definitions of *lawful state space* and *lawful event space* are highlighted in the table to denote that the next highlighted rows contain concepts and definitions subjacent to each concept.

A *variability pattern* represents the different types of variability that can be observed in different products in a particular product line. Variability patterns are derived from a series of similarity classes regarding the variability sources. This dissertation describes the variability patterns included in the ontology (Table 3.2) introducing first the definitions of similarity and equivalence, and then explaining the four variability patterns. A more detailed description is available in [AGWH12]. Let $S = \{s_1, s_2, \ldots, s_m\}$ be a set of elements belonging to a product P_1 and $T = \{t_1, t_2, \ldots, t_n\}$ be a set of elements in product P_2.

Definition 3.1 Equivalence. *S is equivalent to T ($S \equiv T$) iff there is a mapping between S and T.*

Definition 3.2 Similarity. *S is similar to T with respect to an equivalence subset p, denoted as $S \cong_p T$, iff there exists S', T' such as $S' \subset S$ and $T' \subset T$, then $S' \equiv T'$. In other words, the concept of similarity refers to elements that are common to products in a product line.*

Definition 3.3 Full Similarity One-Side. *Two products are fully similar one-side when they satisfy the similarity relation and an equivalence can be established w.r.t. subsets of S or T. Let S', T' be two subsets, $S' \subset S$ and $T' \subset T$, then either $S' \equiv T$ or $T' \equiv S$.*

Definition 3.4 Partial Similarity. *Two products are similar when they satisfy the similarity relation.*

Definition 3.5 Dissimilarity. *Two products are completely dissimilar if no similarity relation can be established.*

Definition 3.6 Ordering. *Variability regarding ordering appears when two products S, T have a similarity relation but they are dissimilar with respect to an ordering relation. Thus, there exists the ordered sets S', T' such as $S' \subset S$ and $T' \subset T$ and $S' \equiv T'$ but S' and T' are dissimilar with respect to their order.*

Table 3.2: Summary of the Ontology [AGWH12]

Concepts			Definition
Variability source	Structure	Things	A thing is an elementary unit. The real-world is made up of things. A composite thing may be made up of other things (composite or primitive).
		Properties	Things possess properties. A property is modeled via a function that maps the thing to some value.
		Lawful state space	The lawful state space is the set of **states** of a thing that comply with the **state laws** of the thing.
		State	The state of a thing is the vector of values for all attribute functions of a thing.
		State law	A state law restricts the values of the properties of a thing to a subset that is deemed lawful because of natural laws or human laws. A law is considered a property.
	Process	Lawful event space	The lawful event space is the set of all **events** in a thing that are lawful.
		Event	An event is a change in state of a thing.
		History	It is the chronologically-ordered states that a thing traverses in time.
Variability pattern	Full similarity one-side		Two products S, T are fully similar one-side when they satisfy the similarity relation, and an equivalence relation can be established w.r.t. subsets of S or T.
	Partial similarity		Two products are similar when they satisfy the similarity relation.
	Dissimilarity		Occurs when no similarity relation can be established.
	Ordering variability		Occurs when two products differ by an order relation.

3.4 Design of the Evaluation

The first step in the evaluation of the HLCL expressiveness is to define the specific goal, questions, and associated metrics to answer the research questions by using the Goal-Question-Metric (GQM) approach [BCR94].

The metrics in this experiment are the four measures of potential ontological deficiencies proposed by Recker *et al.* [RRIG09]: the degree of deficit, the degree of excess, the degree of redundancy, and the degree of overload. Table 3.3 presents how to calculate each measure.

Table 3.3: Measures of potential ontological deficiencies [RRIG09]

Metric	Formula	Definition
M1: Degree of Deficit (DoD)	$DoD = \frac{\#not\ mapped\ ontological\ constructs}{\#ontological\ constructs}$	The ratio of ontological constructs that cannot map to any language construct.
M2: Degree of Excess (DoE)	$DoE = \frac{\#not\ mapped\ language\ constructs}{\#language\ constructs}$	The ratio of language constructs that cannot map any ontological construct
M3: Degree of Redundancy (DoR)	$DoR = \frac{\#lang.const.mapping\ the\ same\ ont.const.}{\#language\ constructs}$	The ratio of constructs in the modeling language mapping the same ontological constructs.
M4: Degree of Overlap (DoO)	$DoO = \frac{\#lang.const.mapping\ many\ ont.const.}{\#languageconstructs}$	The ratio of language constructs mapping more than one ontological construct.

3.4.1 Goal and Research Questions

The main goal of this study is *to evaluate the HLCL with respect to its completeness and clarity from the point of view of the expressiveness in the context of an ontological analysis.* The following are the research questions each paired with their correspondent metrics.

Q1. Does HLCL map all the constructs in the ontological model? This question serves to determine the completeness or incompleteness (construct deficit) of the HLCL using the degree of deficit (DoD).

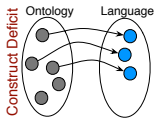
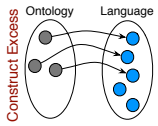
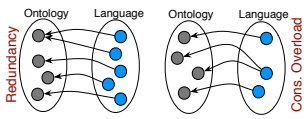
Q2. Are there any HLCL constructs that cannot be mapped into ontological constructs? This question is related to determine if the HLCL has construct excess using the degree of excess (DoE). Therefore, it contributes to elaborate an explanation about the clarity of the language.

Q3. Is the mapping a one-to-one relation? This question serves to determine if the HLCL has construct redundancy and construct overload using the degree of redundancy and degree of overload (DoR, DoO). Thus, together with Q2, Q3 leads to conclude about the clarity of the language.

3.4.2 Hypothesis

The refinement of the stated questions relies on the analysis of three hypotheses, each one with null and alternative forms, related to Recker *et al.*'s metrics, as synthesized in Table 3.4.

Table 3.4: Hypotheses

Question	Null hypothesis	Alternative hypothesis	Defect
Q1	$H1_0$: All ontological constructs were mapped to HLCL constructs. $H1_0 : DoD = 0\%$	$H1_1$: One or more ont. construct cannot be mapped to HLCL constructs. $H1_1 : DoD > 0\%$	
Q2	$H2_0$: All the HLCL constructs were mapped. $H2_0 : DoE = 0\%$	$H2_1$: One or more HLCL const. cannot be mapped to ont. constructs. $H2_1 : DoE > 0\%$	
Q3	$H3_0$: The map is one-to-one. $H3_0 : DoR = 0\% \wedge DoO = 0\%$	$H3_1$: The map is NOT one-to-one. $H3_1 : DoR > 0\% \vee DoO > 0\%$	

3.4.3 Threats to validity

The following section discuss the limitations of this study by elaborating the threats to validity and the strategies used to minimize their effects [KPP+02].

The validity of the results may be affected by the selection of the ontology and the mapping between the language and the ontology. To mitigate the bias in selecting the ontology, the evaluation included a literature review searching for *foundational ontologies* containing domain-independent and variability-related constructs. The following step was to study the concepts in the selected ontologies as well as the purpose and application of the ontology. As a result, the Asadi *et al.*'s ontology was selected. Though this ontology may not be complete as it contains a subset of the constructs from BWW [WW93], the variability patterns included in the ontology describes the possible cases of variability in a product line [AGWH12]. Moreover, one of the conclusions of the evaluation is that similar results can be produced evaluating the HLCL expressiveness using other foundational ontologies along with Asadi *et al.*'s variability patterns. This conclusion considered that the results regarding HLCL's completeness are the product of the difficulties of HLCL to represent ontological constructs related to the sequence of events and temporal constraints. In the case of the HLCL's clarity, overlapping and overload will be present as long as the generic construct *constraints* is used to map all variability constructs in the ontology.

To decrease the threats regarding the mapping between the language and the ontology the analysis was conducted in three steps. First, I as main researcher, separately mapped the HLCL constructs against ontological constructs to create a first mapping draft. Second, Prof. Mazo an I met to discuss and defined a second draft. Third, a last discussion section with both supervisors produced a third version of the mapping. By reaching a consensus at the end of this process, we procured to increase the reliability and validity of the mapping.

3.5 Conduction

The mapping between the HLCL constructs (Table 3.5) and the ontological constructs (Table 3.2) was performed in two steps. The first step consisted on a representation mapping to determine whether and how ontological constructs are represented via a language construct. The second step is to define the interpretation mapping to determine whether and how a grammatical construct stands for a real-world construct and answer the research questions. To close, this section uses the results in the interpretation mapping to measure the potential potential ontological deficiencies in the language applying the Recker *et al.*'s metrics. The following paragraphs describe the representation mapping illustrating it with examples of constraints in the running example: PAS product line (cf, Section 3.1.1).

Table 3.5: Core constructs of the High-Level Constraint Language.

Construct	Definition
Variables	Represent product line elements. A variable has a domain, and it is paired with a value at a given time.
Domain	The domain of variables can be Boolean, integer, interval, enumeration or string.
Values	Represent the elements in the domains.
Constraint	Constraints are relations over a set of variables $C(x_1, x_2, \ldots, x_n)$ producing sets of values $\{v_1, v_2, \ldots, v_n\}$ where v_i is assigned to each variable x_i

3.5.1 Representation mapping

The description of the representation mapping follows the separation of concerns in Asadi *et al.*'s ontology. Then, the description presents first the mapping of the sources of variability followed by the mapping of variability patterns.

Mapping the Sources of Variability.

Things and **properties** were mapped to variables in HLCL. The ontological model defines things as elementary units that have properties. Those properties represent a particular characteristic of a thing. In HLCL, elements in a product line (e.g., feature, requirement, design fragment, component or any other reusable artifact) are represented by variables associated with Boolean domains. The information related to structural elements (things) are the attributes. These attributes are represented using variables with domains of different types, regarding the possible values of the attribute (attributes, values [BSRC10]). In this example, each component in the PAS is mapped to Boolean variables ranging in $\{0, 1\}$ and the attribute Size representing the size of the internal memory is mapped to a variable with domain $\{0, 2, 4, 8, 16, 32\}$. For each attribute, a constraint in the form $(structural\_element > 0) \Leftrightarrow (attribute > 0)$ should be introduced. Therefore, when an element is selected, all its attributes are entailed and vice versa (e.g., C_5 and C_6 in Table 3.1).

In the ontological model, the **lawful state space** is an ontological construct defining the set of states of a thing complying with the state laws of the thing. The mapping considers first the **state** of a thing representing the

possible values of its attributes. In HLCL, the **domain** of a variable represents the set of possible values for such variable. Thus, the state of a thing maps to the set containing the domains of the variable(s) used for representing a thing and its attributes. Second, the mapping considers the **state law** that is a rule that restricts the values of the attributes of a thing. In HLCL, **constraints** are expressions that represent rules restricting the domains of variables. Hence, the state laws are mapped to the constraints over the variables representing the state of a particular thing. Finally, the conclusion is that the **lawful state space** can be mapped to the set of values (in the domain) agreeing with the constraints in the model. For instance, consider the attribute `Cores` in the example in Table 3.1 its domain (state) is defined as the integers in the interval $[0, 7]$. However, considering C_{10} and the domain of the attribute `Size`, the lawful state space for `Cores` is the set $\{0, 1, 2, 3, 4, 5\}$.

The mapping of the **lawful event space** is based on the notion that constraints are not just as the rules in the domain of a system but also as agents in the computational model of the Concurrent Constraints Programming (CCP). CCP is a model for specifying concurrent systems in terms of constraints proposed by Saraswat in [SR90]. In this model, constraints represent partial information about the shared variables of the system that resides in a *store*. This store can be accessed by agents with two basic operations: *ask* and *tell*. Figure 3.2 shows the classic example of four agents interacting with the store.

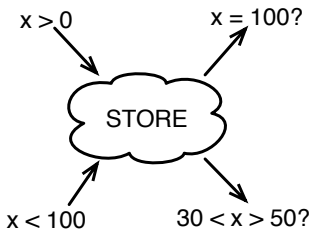


Figure 3.2: A CCP Store Accessed by four agents

The agents at the left tell the store that the variable x will be instanced between the values 0 and 100. When the agent at the top-right asks if x is equal to 100, the answer will be no, but when the fourth agent asks if x is between 30 and 50, it will be blocked because there is not enough information to answer that. The agent will wait until some other agent tells the store something else about the variable.

Now, for the mapping, recall the **lawful event space** as the set of all events in a thing that are lawful (with respect to the state laws). The mapping considers that (1) an **event** is defined as a change in the state of a thing that can be internal or external, (2) states were mapped to HLCL domains in previous mappings. Thus, events are mapped to **constraints** because they are the HLCL constructs that continually produce changes in the domain of variables. More particularly, the mapping considers constraints that trigger other constraints like C_5, C_6, and C_{10} in the example in Table 3.1. In these constraints, the selection of the memory (`Memory` $= 1$) triggers other constraints that will change the domain of variables `Size`, `Processor`, and `Cores`. Now, considering that the lawful state space is represented by sets of values satisfying the constraints, the **lawful event space** is mapped to the constraints in the HLCL model.

The mapping determined a lack of representation for the ontological construct **history**. Asadi *et al.* ontology defines history as the chronologically-ordered states that a thing traverses in time. In HLCL as in other

constraint languages based in CCP [SR90] there exist two moments: first, when the problem is specified and all the possible values in the domain of a variable are defined; second, when the solver determines which values satisfy all the constraints in the problem. Then, a value associated with a variable does not change in time. Therefore, HLCL does not have constructs to specify the sequence of changes in the values assigned to variables.

Table 3.6 presents a summary of the representation mapping of the sources of variability following the format of Asadi's *et al.* ontology.

Table 3.6: Representation mapping between ontological constructs and HLCL constructs.

Concepts			Mapping - Rationale
Variability source	Structure	Things	**Boolean Variables** - Elements in a product line (e.g., feature, requirement, design fragment, component or any other reusable artifact) are represented by variables associated with Boolean domains.
		Properties	**Variables** - The information related to things (e.g., attribute) are represented using variables with domains ranging in the possible values of the attribute. An element is linked to its attributes by a constraint in the form $(structural\_element > 0) \Leftrightarrow (attribute > 0)$.
		Lawful state space	**Values** (in the domain) that respect the constraints in the model.
		State	**Domains** of variables representing attributes. This mapping considers that the state is the set of values for an attribute.
		State law	**Constraints** - State laws as the rule that restricts the values of the attributes of a thing are mapped to the constraints over the variables representing the attributes of a particular thing.
	Process	Lawful event space	**Constraints** over the variables to representing the state of a thing.
		Event	**Constraints** - Considering that events are changes in the state of a thing and states were mapped to domains, events are mapped to constraints that trigger other constraints and produces changes in the domains of elements and attributes.
		History	**No HLCL construct can map this ontological construct.**

Mapping the Variability Patterns.

The variability patterns are observable characteristics of the products in a product line. This mapping considers how the constraints in the HLCL can be used to specify variability relations that generate products where the variability patterns are observable. This mapping included constraints representing variability in previous publications [SMD+11a, MSD+12b] To illustrate this mapping, Table 3.7 shows a subset of the valid products in the running example and particular instances of constraints mapping variability patterns in Table 3.8.

Full Similarity One-Side. This pattern can be mapped to constraints in HLCL used for representing optional relations [KCH+90]. Within an optional relation, it is possible to find products including an element, or not. Then, the use of optional relations causes the inclusion of two or more products with the full similarity one-side property in the set of solutions. For example, consider products P_1 and P_3 in Table 3.7 and $P'_3 \subset P_3$ where $P'_3 =$ `{Processor, Cores=1, Memory, Size=2, Sensors, SpeedSensors}`. Then all items in P_1 are also in P'_3, then $P'_3 \equiv P_1$. Optional relations can be represented with boolean and integer constraints. For instance, in

Table 3.7: Examples of valid products in the Movement Control System (PAS) product line.

P_1 :	`{Processor, Cores=1, Memory, Size=2, Sensors, SpeedSensors}`
P_2 :	`{Sensors, PositionSensors, Feedback, Visual, Audio}`
P_3 :	`{Processor, Cores=1, Memory, Size=2, Sensors, PositionSensors, Feedback, Audio, Vibration}`
P_4 :	`{Processor, Cores=3, Memory, Size=8, Sensors, SpeedSensors}`
P_5 :	`{Processor, Cores=1, Memory, Size=2, Sensors, SpeedSensors, Feedback, Visual}`

the PAS example, C_2 to C_4 are optional relations. Other examples of constraints used to represent optional relations in HLCL are in Table 3.8.

Table 3.8: Constraints mapping variability patterns.

Variability Pattern	Constraint	HLCL Constructs	Semantics
Full similarity one-side	(1)$C \Rightarrow P$	**Constraints** with Boolean domains, and logic operators.	If P is selected then C may be selected, but if C is present, then P is present too.
	(2)$C \leq P$	**Constraints** with integer domains, arithmetic operators.	
Partial similarity	(1)$P \Leftrightarrow C_1 \vee \cdots \vee C_n$	**Constraints** with Boolean domains, and logic operators.	If P is selected then one or more C_i are selected.
	(2)$(C_1 \Rightarrow P) \wedge \cdots \wedge (C_n \Rightarrow P)$ $\wedge\ P \geq 1 \Rightarrow C_1 + \cdots + C_n \geq m$ $\wedge\ P \geq 1 \Rightarrow C_1 + \cdots + C_n \leq n$	**Constraints** with integer domains, and Boolean, arithmetic operators.	If P is selected then at least m and at most n C_i are selected.
Dissimilarity	$(C_1 \Rightarrow P) \wedge \ldots \wedge (C_n \Rightarrow P)$ $\wedge\ P \geq 1 \Rightarrow C_1 + \cdots + C_n \geq 1$ $\wedge\ P \geq 1 \Rightarrow C_1 + \cdots + C_n \leq 1$	**Constraints** with integer domains, and logic, arithmetic operators.	If P is selected then, one or zero C_i are also selected.
Ordering variability	No HLCL construct can map this ontological construct.		

Partial Similarity. This pattern can be mapped to constraints in HLCL used for representing variability relations that produce products sharing common elements. The most common variability relations with this characteristic are OR-relations and group cardinality $\langle m, n \rangle$ relations [CHE05]. Table 3.8 shows how to represent OR and group cardinality in HLCL. For instance, consider C_8 that represents a group with cardinality $\langle 1, 2 \rangle$ and C_9 representing a group with cardinality $\langle 1, * \rangle$. These constraints produce valid products such as P_3 and P_5 exhibiting partial similarity where they share some items `{Processor, Cores=1, Memory, Size=2, Sensors, Feedback}`. Note that it is not possible to produce a full similarity with P_3 and P_5 (not even a full similarity one-side).

Table 3.9: Interpretation mapping between ontological constructs and HLCL constructs.

Constructs		Ontology								
		Variability Sources					Variability Patterns			
		Things	Properties	Lawful state space	Lawful event space	History	Full similarity one-side	Partial similarity	Dissimilarity	Ordering variability
HLCL	Variables	•	•							
	Domains			•						
	Values			•						
	Constraints				•		•	•	•	

Dissimilarity. To produce completely dissimilar products, the variability model should not contain constraints that enforces the inclusion of an item in all products of the product line (e.g., mandatory). Then, this mapping does not consider the core items in the product line. The *dissimilarity pattern* can be mapped to constraints in HLCL used for representing variability relations such as alternative (XOR) [KCH+90], or group cardinality $\langle 1, 1 \rangle$ [CHE05]. These relations produce products without common elements. For instance, without considering the `Sensors` that are always selected, products P_2 and P_4 do not have common elements as a consequence of the optional relations C_2, C_3 and the alternative relation in C_8. Table 3.8 shows how to represent alternative and group cardinality $\langle 1, 1 \rangle$ in HLCL.

Ordering This variability pattern cannot be mapped in HLCL because there are no constraints to determine the order of the selection of values for variables. In addition, under the concurrent constraint programming model, it is not possible to establish an ordering for the application of constraints [SR90].

3.5.2 Interpretation mapping

The interpretation mapping determines whether and how a grammatical construct stands for a real-world construct. Accordingly, Table 3.9 presents the interpretation mapping. Rows in Table 3.9 represent the HLCL constructs, and columns represent the ontological constructs. A bullet is depicted when the HLCL construct in the row maps the ontological construct in the column.

3.5.3 Measuring the potential ontological deficiencies

The potential ontological deficiencies of the `HLCL` are measured by calculating the four metrics proposed by Recker *et al.* [RRIG09] presented in Table 3.3. The calculations as follows:

M1: Degree of Deficit (DoD). M1 is calculated by dividing the number of not mapped ontological constructs over the total number of ontological constructs. Thus the *DoD* is calculated as follows:

$$DoD = \frac{\#not\ mapped\ ontological\ constructs}{\#ontological\ constructs} = 0.22$$

Where the number of ontological constructs is nine and the number of not mapped ontological constructs is two. The latter because ontological constructs *history* and *variability ordering* were not mapped to any language construct. In consequence, HLCL exhibits a 22% of degree of deficit. Following Recker *et al.*'s proposal, the complement of the DoD represents the level of ontological completeness. Then, HLCL's completeness level is 78%.

M2: Degree of Excess (DoE). M2 is calculated by dividing the number of not mapped language constructs over the total number of language constructs. Thus the *DoE* is calculated as follows:

$$DoE = \frac{\#not\ mapped\ language\ constructs}{\#language\ constructs} = 0$$

All the language constructs were mapped. Then, HLCL has zero degree of excess (DoE 0%).

M3: Degree of Redundancy (DoR). To calculate M3 we divide the number of language constructs mapping the same ontological construct over the total number of language constructs. The DoR is calculated as follows:

$$DoR = \frac{\#lang.const.mapping\ the\ same\ ont.const.}{\#language\ constructs} = 0.5$$

Where the number of language constructs is four and the number of language constructs mapping the same ontological construct is two. Table3.9 shows that both language constructs, *Domains* and *Values*, map the ontological construct *Lawful state space*. Then, the HLCL's degree of redundancy is 50%.

M4: Degree of Overlap (DoO). M4 is calculated by dividing the number of language constructs mapping more than one ontological construct over the total number of language constructs. The DoR is calculated as follows:

$$DoO = \frac{\#lang.const.mapping\ many\ ont.const.}{\#languageconstructs} = 0.5$$

Where the number of language constructs is four and the number of language constructs mapping more than one ontological construct is two. As depicted in Table3.9, the language constructs: *Variables* and *Constraints* map more than one ontological construct. Then, HLCL's degree of overlap is 50%.

3.5.4 Results

Q1: Does HLCL map all the constructs in the ontological model?

Both the representation and interpretation mapping showed a construct deficit for representing the ontological constructs: history and ordering (highlighted columns in Table 3.9). Moreover, HLCL does not support the design of product line models where explicit consideration is given to sequence and order in the product line.

Accordingly, HLCL users will encounter difficulties in meeting the potential need for explicit represent constraints such as: "element E_1 must be selected before E_2". Regarding the results of Asadi *et al.* in [AGWH12], a similar deficiency was also found to exist in FMs and OVM. Consequently, neither HLCL, FMs nor OVM can represent history and ordering.

Q2: Are there any HLCL constructs that cannot be mapped into ontological constructs?

All constructs in HLCL were mapped to the elements in the ontology proposed by Asadi *et al.* [AGWH12]. Hence, HLCL does not present construct excess.

Q3: Is the set of mapped constructs a one-to-one relation?

The mapping of ontological constructs to language constructs is not one-to-one. Table 3.9 shows that it is not possible to have a one-to-one mapping considering the difference in the number of constructs (HLCL has four constructs, the ontology has nine constructs). In consequence, the absence of a one-to-one mapping causes two defects: (1) there is one ontological construct mapping to more than one HLCL constructs (construct redundancy), and (2) there are HLCL constructs mapped to different ontological constructs (construct overload). These two defects are also observable in Table 3.9. First, columns in Table 3.9 serve to identify which ontological constructs are mapped to more than one HLCL construct. Thus, columns with more than one bullet are instances of construct redundancy. In the table, one instance of construct redundancy is observable, the mapping (lawful state space $\rightarrow$ domains, values). Second, rows in Table 3.9 are used to find which HLCL constructs are mapped to more than one ontological construct. Therefore, rows with more than one bullet are instances of construct overload. As seen in the table, two of the four HLCL constructs are involved in more than one mapping: variables and constraints.

Ontological Completeness and Clarity

To discuss about the ontological completeness and clarity, recall the metrics measuring the potential ontological deficiencies calculated in Section 3.5.3. The Recker's *et al.* Degree of Deficit (DoD), is the measure usually used to conclude about the level of ontological completeness in a conceptual modeling language. Under this idea, the lower the DoD, the higher the level of ontological completeness. HLCL exhibits a 22% of degree of deficit. Therefore, HLCL completeness level is 78%. This result means that HLCL closely represents the general principles of variability under the Asadi *et al.* ontological framework. To evaluate the ontological clarity of HLCL, the Degrees of Excess (DoE), Redundancy (DoR) and Overlap (DoO) were calculated. HLCL exhibits a low degree of excess (DoE 0%), and medium degrees of redundancy and overload (DoR, DoO 50%). On the one hand, a low DoE is a desirable situation as it prevents user confusion due to the need to ascribe meaning to constructs that do not appear to have real-world meaning. On the other hand, the levels of redundancy and overload indicate that HLCL might be unclear and will produce potentially ambiguous representations of real-world domains.

3.6 Lessons Learned

The analysis of the results obtained in this evaluation of the HLCL under the theory of ontological expressiveness contributes to taking forward the discussion of HLCL's expressiveness. The results of the evaluation should be analyzed from two different perspectives: clarity and completeness.

3.6.1 Clarity vs Abstraction

HLCL presents a medium level of clarity due to its levels of redundancy and overload. This redundancy and overload levels depend on the number of constructs in HLCL and especially in the repeated use of constraints for mapping variability patterns. The *constraint* construct is a generic construct that represents a set of expressions. The mapping presented in Section 3.5.1 includes particular instances of constraints to explicitly demonstrate how HLCL constructs represent variability patterns in the ontology. To enhance the clarity of the new language, it needs a set of constructs containing expressions mapping frequently used variability relations. These new constructs would be considered syntactic sugar as they can be removed without any effect on the expressive power of the language.

3.6.2 Ontological (in)Completeness

Ontological incompleteness arises because it is not possible to map any HLCL construct with the ontological constructs related to time: history and ordering. A similar observation was reported by Asadi *et al.* after analyzing FMs and OVMs [AGWH12]. In their study, Asadi *et al.* concluded that both languages lack variability completeness as they do not have any construct for representing order. This conclusion is not surprising given that the formalisms associated with these variability languages (e.g., first-order logic and concurrent constraint programming [BSRC10]) do not model time. Indeed, the lack of expressiveness concerning the notion of time is not inherent only to these three languages. To the best of our knowledge, there is no product line notation including constructs for modeling time in product lines. Therefore, the gap in the HLCL's expressiveness demonstrated in this study, reflects a gap in the state of the art of the product line notations. Consequently, the inclusion of time as a native concept in variability languages should be considered as a challenge in the design of a new variability language.

3.6.3 What About Time for Variability Modeling?

The notion of time in computational models represents the sequence of changes in the state of a system. Time can be used in product line notations to specify variability in process or behavior and also variation between product releases. To address these, notations require sophisticated elements able to represent temporal constraints aiming to (1) enhance variability, (2) schedule changes, and (3) sequence constraints. Temporal constraints *enhance variability* languages by allowing to represent preferences regarding the moment of activation of an element in a product line including constraints such as "element A is activated before/after the activation of element B", "in the next time unit, element A is activated", or "element A is activated after three units of time". These temporal and scheduling constraints are particularly relevant to Dynamic Software Product Lines (DSPLs). The goal of the DSPL is to build systems that dynamically adapt themselves to fluctuations in user needs, environmental

conditions, and resource constraints at runtime. In this context, temporal constraints might be useful for including rules to schedule reconfigurations (adaptations). The inclusion of temporal constraints might enable the use of constraints such as: "the reconfiguration starts at time x", "the reconfiguration occurs during event E", "a reconfiguration will occur eventually". *Sequence constraints* are useful to perform staged configuration where it is necessary to produce a series of intermediate configurations compelling to a collection of requirements. For instance, Burdek *et al.* in [BLL+13] include temporal constraints to perform staged configuration to determine an order relationship between configuration stages. In their work, Burdek *et al.* include time constraints in a feature-based notation.

3.6.4 The Theoretical Evaluation Framework

Many works have applied the theory of ontological expressiveness to evaluate conceptual modeling languages such as the entity-relation model [SMN+10]; UML [BJM08]; the $i^\star$ language [GFG12]; and BPMN [RRIG09] to mention a few. In the domain of product lines, there are two well-known works [RBSW11, AGWH12] using the theory of ontological expressiveness, both of them using the BWW ontology [WW93] as a basis and extending it to accomplish their purposes. First, Reinhartz-Berger *et al.* [RBSW11] included a set of constructs to analyze process variability and later to determine variability in terms of software behavior. Second, Asadi *et al.* [AGWH12] included a collection of variability patterns aiming to provide a framework for evaluating the ontological expressiveness of variability modeling languages. In their work, Asadi *et al.* use their framework to evaluate two variability languages: FMs and OVM.

The evaluation framework conformed to conduct the ontological evaluation is a collateral result of the design and conduction of the ontological evaluation presented in this chapter. This evaluation applies the theory of ontological expressiveness to provide a theoretical analysis of HLCL as a variability modeling language in order to determine its ability to represent variability in real-world domain models. To the best of our knowledge, there is no evaluation of the expressiveness of a language that abstracts product line constraints as HLCL does. The particulars of the evaluation and the generality of the research questions tackled in this evaluation resulted in the composition of different elements that could be reused to evaluate other variability modeling languages. For instance, Achtaich *et al.* [ARS+19] apply the evaluation framework defined in this chapter to assess the expressiveness of their modeling language for defining variability in the context of self-adaptive systems. Moreover, this framework will be used in Chapter 5 to evaluate the variability language proposed in this thesis.

3.7 Summary of the Practical Evaluation

The last step in the evaluation of the `HLCL` framework consisted in actively getting involved on the engineering team supporting the VariaMos tool-suite. This participation aimed to get the insight and experience the problematic situation associated to design software solutions for supporting variability modeling and analysis.

The VariaMos tool-suite [MMFR+15] is a variability management tool which provides tooling support for modeling and reasoning in different variability modeling languages. This tool embeds the `HLCL` framework a joint contribution of many research works [DS08, SMD+11a, MSD11, MFTR+15]developed at the *Centre de*

Recherche en Informatique. This framework exploits the idea of relying on an intermediate language unifying existing notations to provide genericity to the methods, techniques, and tools used for modeling, analysis, and configuration. Figure 3.3 depicts a conceptual model and refinement of the HLCL framework derived from theoretical background and the experience of work as part of the engineering team.

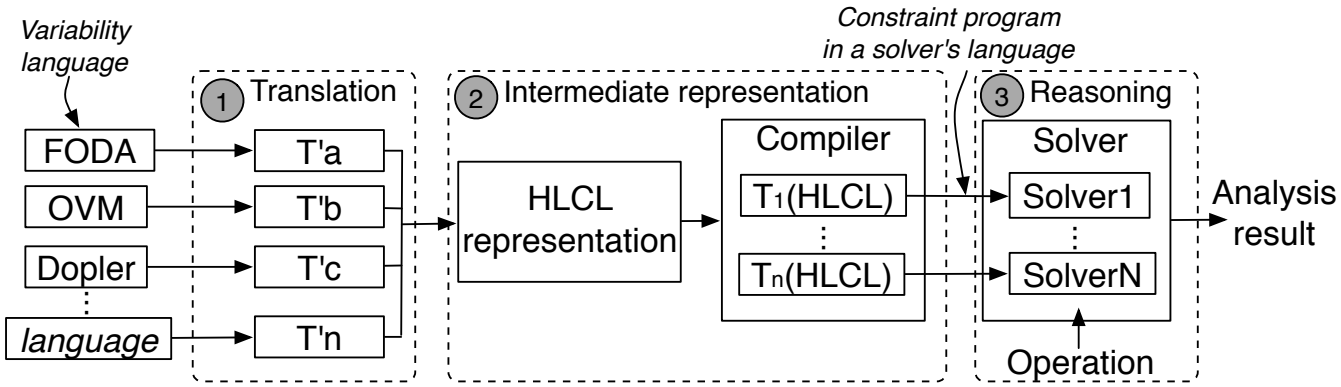


Figure 3.3: The HLCL framework.

The inclusion of the *intermediate representation* causes the inclusion of an extra-step in the transformation framework as shown in Figure 3.3. First, a collection of rules are applied to encode variability models into HLCL representations using the rules in [SMD+11a, MSD+12b]. Second, the variability model encoded in the HLCL generic representation goes through a compilation process to produce a solver compatible representation. This step is crucial to support different solvers for reasoning purposes. Finally, the reasoning task is completed by executing the query using a constraint solver.

The intermediate representation approach in the HLCL framework is useful for integrating different variability languages in one single tool. The VariaMos tool-suite provides a graphical interface to describe variability of a single system using different views and more than one notation. These different views are later integrated in a single model represented in HLCL to support reasoning providing several analysis and verification operations [MMFR+15]. However, the benefits of the HLCL framework for supporting different notations and solvers do not compensate for the drawbacks regarding the following key factors.

1. The intermediate language exhibits incompleteness, lack of usability, and readability. At some point, to use this language resembled replacing a programming language by assembly language: regardless of its benefits, to work with large scale assembly programs without a higher level, more abstract language is an unfeasible task. Additionally, from the point of view of the implementation, the approach to provide an in-house abstract representation limits the constraints, functions, and operators available to represent variability models. Therefore, limiting the expressiveness level of the modeling language.

2. The intermediate representation of variability models using HLCL does not provide full-solver independence. Indeed, the usage of multiple solvers in the HLCL is unpractical because it requires an extra transformation step to encode HLCL representations into solver-readable code, as shown in Figure 3.3. Thus, the inclusion of new solvers involves the definition and implementation of new transformation rules.

3. The intermediate representation of variability models using HLCL does not solve the issues regarding

the portability and sharing of models between tools. Variability models initially described in different notations and later encoded into HLCL representations are useful for being analyzed in an integrated way in the same tool [DTS+14, MFTR+15]. However, sharing models encoded in HLCL between different tools is unpractical because of the loss of variability information in the transformation.

4. The extensibility of the tool implementing this framework, namely the inclusion of new languages and new solvers, requires the definition and implementation of new transformation rules. The addition of a new modeling language in the tool involves the definition of the rules for encoding variability languages into HLCL representations. Additionally, the inclusion of a new solver demands the definition of the rules for obtaining a solver-readable encoding. Then, the usage of an intermediate representation does not solve the reimplementation of the transformation chain.

The inclusion of an intermediate representation in the transformation chain provides an extra level of abstraction for designing a tool supporting different modeling languages and solving tools. In practice, the `HLCL` framework does not ease the difficulties to share and port variability models between tools. Thus, VariaMos and any implementation of this framework need external parsers to import models from other tools. Furthermore, the `HLCL` framework replicates the portability issues inside the intermediate representation layer as a consequence of the in-house definition and implementation of the HLCL. HLCL is a subset of constraint programming containing the operators and expressions employed to encode variability models. Therefore, a standard representation of constraint programming may be a better candidate to encode models as constraints in a solver-independent way.

3.8 Towards the Coffee framework

The results of the evaluations showed that the benefits of the `HLCL` framework for supporting different notations and solvers do not solve the problematic situation addressed in this thesis. Moreover, these results point to a new direction on the objective to use intermediate languages to ease the interoperability of variability management tools by designing a constraint-based framework supporting an *expressive* variability language and a *flexible* automated analysis mechanism.

The results of the evaluation showed that there exist two different concerns in the goal of providing a constraint-based framework for variability modeling and reasoning. On the one hand, there exist concerns regarding the variability modeling. These concerns are centered on how variability languages should be designed to provide enough expressiveness and abstraction to the modeler. On the other side, there exist concerns related to constraint programming and its connection to the variability language for providing a reasoning mechanism encompassing the expressiveness of the modeling language in a solver-independent way. At this point in the research, two questions pointing two different directions were raised by the results of this study: *is it appropriate to pursue the usage of a generic constraint-based language as intermediate variability representation?* or *should analysis concerns be treated separately from the modeling concerns?* even if they are so intrinsically related? We opted to continue this work in the second direction and design the Coffee framework with two intermediate representations, one for variability and one for analysis.

3.8.1 Coffee's Overview

Coffee is a constraint-based framework that contributes to ease the persistent issues concerning variability modeling and reasoning about variability models.

This proposal exploits the usage of intermediate representations of variability models that can be portable and shared by different tools. This intermediate representation, accompanied by a flexible transformation mechanism, enables Coffee to exploit the advantages of different solving paradigms and solvers.

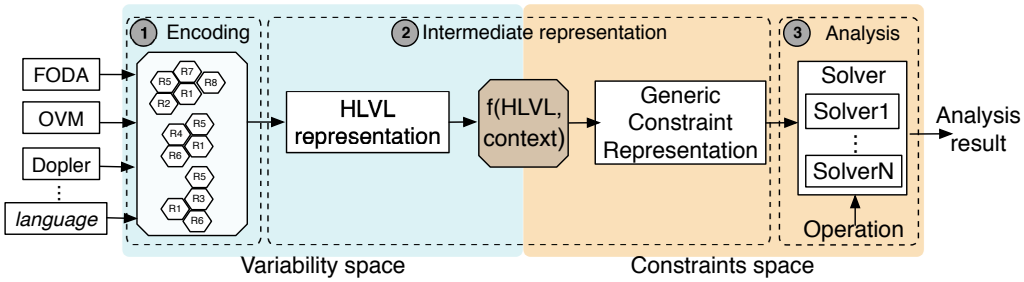


Figure 3.4: Overview of the Coffee Framework

This section presents an overview of the constraint-based framework for supporting expressive variability modeling, and flexible variability analysis the Coffee framework.

Figure 3.4 presents the conceptual model of Coffee. The design of this framework continued the premise of including intermediate representations. Similarly to the HLCL framework, Coffee includes an intermediate representation layer and an extra step in the transformation chain. What is different in Coffee is that this additional layer encodes variability models using two intermediate representations: one variability centered representation for modeling and a second constraint centered representation for automated analysis. This separation of languages answers to the separation between modeling and analysis concerns reported in the evaluation. Figure 3.4 illustrates how each part of the framework belongs either to the *variability space*, *i.e.,* colored in blue, or to the *constraints space*, *i.e.,* colored in orange. The only exception is a *transition step*, colored in brown, interconnecting both spaces. This transition space represents the flexible transformation mechanism to encode variability models into constraint satisfaction problems. The following subsections describe the variability space, the constraint space, and the transition step.

3.8.2 The Variability Space

The *variability space* encircles the elements of the framework designed to address the issues regarding the the lack of standards and its impact in the low portability of variability models and the interoperability issues between variability management tools. This space encompasses the High-Level Variability Language (HLVL), the intermediate representation designed to fulfill the ontological expressiveness and to cope with basic and complex concepts from variability modeling languages, and to work with/for/in combination with most languages and tools. HLVL is a textual variability modeling language, with features that makes it a candidate to be used as exchange format among variability management tools. HLVL is the major contribution of Coffee and become the angular element in the variability space. Chapter 4 details the design of HLVL.

As shown in Figure 3.4, the variability space contains two parts.

1. An *Encoding* step, also called *models' parsing* defining the transformation patterns to obtain an HLVL representation of variability models written in machine-readable format.
2. The *HLVL representation* containing the language's infrastructure proposed for specifying variability models in HLVL (*e.g.,* syntax, editor, etc.).

Parsing other languages into HLVL models

Definition 3.7 Semantic equivalence. *Semantic equivalence is when we have two variability models in two languages, but the models mean the same in terms of configurations.*

3.8.3 The Transition Step

The transition step between the variability space and the constraints space consist in the transformation to encode `HLVL` models into a Generic Constraint Representation (GCR). The transformation rules and the mapping function $f(HLVL, context)$ are illustrated as a box in Figure 3.4. This transition step contains the rules conforming the `HLVL`'s operational semantics defined in Chapter 4.

3.8.4 The Constraints Space

The *constraints space* contains the elements in Coffee to support a solver-independent reasoning mechanism. The design of the constraints space develops the idea that a *generic* solver-independent intermediate representation of variability models favors the usage of solvers as plug&play tools. Several approaches relates the usage of one or many *off-the-shelf* solvers to support the reasoning in their tools as if they are interchangeable. However, as it has been discussed before, there exists a strong dependency between the elements in the transformation framework. Thus, new rules and parsers must be designed and implemented to support new solvers.

To address this issue, Coffee uses a standard representation of constraint programming compatible with several solvers able to be extended externally, *i.e.,* without including transformation steps in the chain. Figure 3.4, depicts the following elements in the constraints space:

1. The *Constraint Generic Representation* layer containing the infrastructure for processing the variability models represented as constraint problems. MiniZinc [NSB+07] is the technology selected to provide an implementation to the CGR. MiniZinc is a solver-independent modeling language supporting most of the solvers used for reasoning about variability models.
2. The *Solving* layer is used to execute solvers compatible with the MiniZinc tool-chain and therefore, able to read MiniZinc representations of variability models. In addition, this layer is in charge of parsing the output obtained from the solver to produce the analysis results.

3.9 Summary

This chapter reported the results of the first stage in the research presented in this dissertation and the contributions resulting from this stage.

First, Sections 3.2, 3.3, and 3.4 present the evaluation framework to measure the expressiveness of variability modeling languages. This framework is grounded on the theory of ontological expressiveness [WW93], using a foundational ontology for variability languages combining the works of Asadi *et al.* [AGWH12] and Reinhartz-Berger *et al.* [RBSW11]. In addition, the evaluation process follows the GQM approach considering the metrics to evaluate ontological expressiveness and clarity in a conceptual modeling language developed in the works by Recker *et al.* [RRIG09].

Second, Section 5.1.2 presents the application of the evaluation framework to the High-Level Constraint Language, the results obtained, the discussion and conclusions. These results point to new challenges in the research presented in this thesis and pointed a new direction. Additionally, Section 4.5 summarizes the experience on the engineering a variability management tool, and signals the drawbacks discussed and shared among the community. From this point on, the concerns regarding variability modeling and variability analysis were treated independently in the thesis.

Finally, Section 3.8 presents the conceptual model of the Coffee framework, the original proposal developed in this thesis. Coffee solves the interoperability among modeling tools introducing an expressive textual variability modeling language called the High-Level Variability Language. Also, in Coffee the strong coupling and lack of flexibility in the transformation framework are solved by introducing a Generic Constraint Representation together with a context-aware transformation framework.

The next chapter details the components of the Coffee framework starting from the variability language.

Chapter 4

Variability Modeling and Variability Analysis in Coffee

This chapter shares content with the paper *The High-Level Variability Language: an ontological approach* [VMS19]

The previous chapter presented the process and results produced in the first stage of this research and closed with the introduction of Coffee, a framework designed to solve the issues in variability modeling and analysis using intermediate representations. The framework's design complies with the separation of concerns regarding variability modeling and variability analysis. This separation of concerns is reflected in the framework's structural division among the *variability* and *constraint* spaces.

This chapter presents the elements within the variability and constraint spaces and how these elements are orchestrated in Coffee to support variability modeling and variability analysis. The following sections present (1) the characteristics and formal definition of the High-Level Variability language; and (2) the HLVL's operational semantics as the core for supporting variability analysis in a flexible and multi-solver fashion.

4.1 Motivation and Challenges

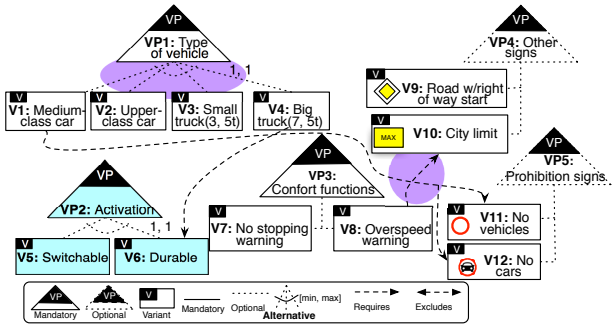
Modeling variability and analyzing variability models are entwined activities. Variability modeling requires automated analysis to produce high-quality, defect-free, and maintainable models. However, as evidenced in Chapter 3, variability modeling and analysis concerns should be treated separately. Accordingly, the following subsections present the challenges on variability modeling and variability analysis considering the separation of variability modeling and analysis concerns.

4.1.1 Variability Modeling Concerns

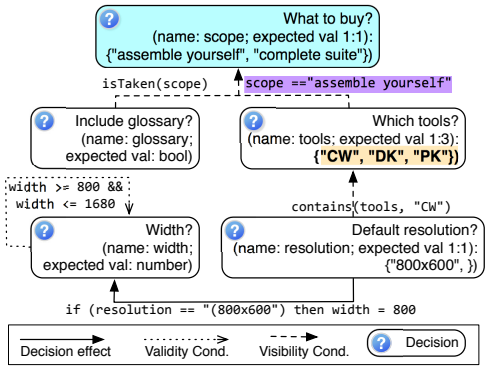
Variability languages enable the modeler to answer two questions about the system to be modeled: *what does vary?* and *how does it vary?* [PBvdL05]. These languages provide a collection of *constructs* enabling the modeler (1) to identify and document the variable items; (2) to identify the set of possible options or variants associated to variable items in the system; and (3) to define the rules for determining how variable items can be combined into new configurations.

Different modeling approaches have been proposed since the introduction of FODA, such as, variation-point-oriented languages [PBvdL05], decision-oriented languages [DGR11], constraint-oriented languages [SMDD10] and industrial languages (*e.g.,* Kconfig [ZC], Gears [Kru07]). These proposals have contributed to a universe of languages, notations, transformations, and tools supporting the creation of variability models. Then, variability modeling relies upon existing domain-specific languages and modeling tools. These tools are developed and taught in-house and frequently are used only by the few people associated with the development team, as we ourselves had experienced with the development of the VariaMos tool suite [MMFR+15].

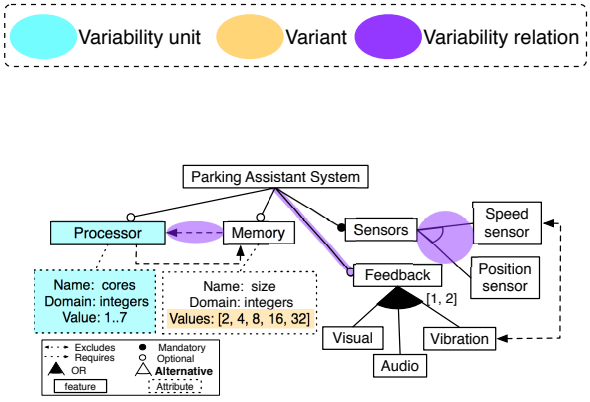
This diversity of modeling paradigms, languages, tools, representations (graphical or textual) is the main cause of the lack of standards for modeling variability, lack of portability of variability models, interoperability between modeling tools, and the difficulty task of share models for comparisons and benchmarking. However, Variability modeling languages are neither completely different nor completely the same. Let's take a look to Figure 4.1 containing four variability models written in different languages: *FODA* [KCH+90], *Dopler* [DGR11], *OVM* [PBvdL05], and TVL [CBH11].



(a) Orthogonal Variability Model (OVM) [PBvdL05]



(b) Dopler Modeling Language (DoplerML) [DGR11]



(c) FODA [KCH+90] with the extension proposed in [BTRC05a].

```
01 root Voting {
02  enum defaultVoteValue in {yes, no};
03  (defaultVoteValue == yes) -> Yes;
04  (defaultVoteValue == no) -> No;
05 group allOF {, voter}},
07     VoteValues group [2..*] {
06     Encoder {group someOf {manager, voter}},
08       Yes,
09       No,
10       Abstain,
11       NotYetEncoded,
12       NotVote,
13     }
14 }
15 }
```

(d) Textual Variablity Language TVL [CBH11].

Figure 4.1: Variability models using different languages.

Similarities. All four models contain syntactic elements or constructs (graphical, textual or both) representing the things that vary (*i.e.,* variability units, examples highlighted in blue), the possible choices (*i.e.,* variants, examples highlighted in orange), and the rules for combining these variable items into products (*i.e.,* variability relations, examples highlighted in violet). Moreover, some of these constructs share or have similar semantics. Other constructs even share their names as in FODA and OVM models.

Differences. The models in Figure 4.1 have (1) different modeling paradigms, *i.e.,* feature models, variation-point models, and decision models; (2) different expressiveness levels, *e.g.,* some support boolean and non-boolean information and cardinalities, others do not; (3) different structures, *e.g.,* hierarchical tree-shaped, graph-shaped, and nested blocks; (4) different types of variability units, *e.g.,* associated to Boolean, and non-Boolean variants; (5) heterogeneous rules, *e.g.,* cross-tree constraints, visibility, and validity conditions, and constraints; among others.

Nevertheless, all the variability languages used in the example have been transformed in constraint satisfaction problems to perform analysis, most of them with the same transformation rules. Following the idea that despite the interoperability issues, the diversity in the modeling approaches also causes richer models and dedicated tools, this chapter presents the *High-Level Variability Language (HLVL)*, the proposal developed in this thesis for a unified variability modeling language that supports basic and complex concepts from different variability modeling languages and is able to work with/for/in combination with most languages and tools. To achieve the goals of defining HLVL, this research tackled the following challenges.

CH1. To determine the expressiveness level of the modeling language. Two sources were considered to determine the expressiveness level of HLVL. On the one hand, the glossary of variability modeling concepts surveyed in the literature review and reported in Chapter 2. On the other hand, the criteria for ontological expressiveness from the ontological analysis reported in Chapter 3. With these inputs in consideration, this chapter presents the decisions regarding *which variability concepts* does the HLVL support and *why.*

CH2. The definition of the characteristics of a variability modeling language for being used for modeling variability and for being considered as exchange format. The design of HLVL is centered on the language expressiveness and its capabilities to support most current languages. However, the design of the language raises other questions such as (1)*which modeling paradigms should HLVL support*; (2) *what are the characteristics of the language's concrete syntax*; and (3) *does HLVL meet the scenarios considered in the initiatives to propose a modeling standard* [BC19]?

CH3. The definition of a variability modeling language avoiding implementation-dependency
To avoid implementation-dependency, the semantics of HLVL should present the rules to determine the meaning of a variability model in terms of the set of valid configurations independently of the implementation. This is achieved by the definition of the language formal semantics which is the hallmark of precision and unambiguity and a prerequisite for efficient and safe tool automation [SHTB07]. Thus, this chapter answers the questions about (1) *what does a variability model specified in the HLVL language mean*?; and (2) *what is the approach to represent a formal semantics of the language to unveil the language's behavior of any particular implementation*?.

4.1.2 Variability Analysis Concerns

Similar to other state-of-the-art approaches, variability analysis in Coffee requires the transformation of variability models into logic representations to perform satisfiability questions on solvers. As the representation of variability models in a logic paradigm is crucial to support most variability management tasks [MBC09, TKB+14, BSRC10, GBT+18].

The question, *Which set of transformation rules should be considered to encode variability models into constraint satisfaction problems?* is recurrent when engineering variability management tools. Several sets of rules can be applied to encode variability models for analysis purposes. For instance, the literature review conducted in this thesis, *i.e.,* Chapter 2, describes 25 types of transformation rules. Furthermore, other literature reviews, such as [BSRC10, GBT+18], reference at least 16 different publications containing rules to transform variability models into constraint programs.

The answer to the latter question is: *it depends*, as it happens that the complexity of the logic paradigm and the set of transformation rules *depend on* the expressiveness of the modeling language. This situation is the so-called trade-off between expressiveness and analysis [MP14, BSRC10, EKS13, GBT+18]. Hence, there is no particular way to encode variability models into constraint problems but a collection of transformation rules that may apply regarding the set of variability concepts supported by the language.

Following this idea, dealing with the complexity of the logic paradigm and the transformation rules poses a significant challenge in the design of the automated analysis support in Coffee, since the main characteristic of HLVL is *expressiveness*. Therefore, to provide a *flexible* transformation framework that works with different logic representations and solving tools, the following challenges were added to the ones explained in the previous section.

CH4. To define the logic representation and transformation rules for encoding variability models in HLVL for analysis purposes. Two questions guide the definition of the logic representation and transformation rules: What is the approach to encode HLVL models without impact the expressiveness of the language? Which logic representations and transformations rules are better suited for models in HLVL?

CH5. To define the intermediate representation used for encoding variability models in the selected logic representation. The design of the transformation framework develops the hypothesis to encode variability models using an intermediate representation to enable the multi-solver support. Then, the idea is to replace the HLCL to support different types of solvers and avoid the implementation dependence evidenced in the evaluation. Then, *what intermediate representation for constraint satisfaction problems can be used in the implementation of the transformation framework?*

4.1.3 Examples in this Chapter

This chapter presents three examples to describe the language constructs and to illustrate how HLVL can be used to model characteristics of different variability modeling languages.

The main example is the fictional case of a car's Parking Assistant System (PAS), described in Chapter 3.The PAS is a system that assists drivers for parking their vehicles using sensors and feedback devices for controlling

the speed and correct the car's trajectory. This example describes the basic scenario that will grow as the chapter includes modeling concepts. Then, the extensions are cumulative. Figure 4.2 recalls the definition of the running example and presents a graphical depiction of the example in the graphical FODA [KCH+90] language with the extension proposed by Benavides *et al.* [BTRC05a].

The PAS is composed of a processor, an internal memory slot, some sensors, and optionally some feedback devices. A PAS may contain zero or more speed sensors and position sensors. Feedback can be visual, auditory or vibratory, and a single product can have at most two kinds of feedback. When a speed sensor is included in a configuration, then vibratory feedback must be excluded, and vice versa. The PAS uses a processor that can have one to seven cores. The size of the internal memory is a value in the set $\{2GB, 4GB, 8GB, 16GB, 32GB\}$. The size of the memory depends on the number of cores in a processor, the pair ⟨`cores; memory size`⟩ can have the following values ⟨`0;0`⟩, ⟨`1;2`⟩, ⟨`2;4`⟩, ⟨`3;8`⟩, ⟨`4;16`⟩, ⟨`5;32`⟩.

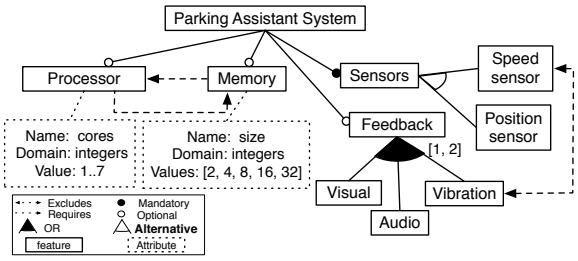


Figure 4.2: Summary of the running example from Chapter 3

Other Illustrating Examples

The examples in Figure 4.1 presented in the motivation to compare variability languages will be used to illustrate how specific constructs of different languages can be represented using HLVL in the following sections.

1. Figure 4.1a depicts an excerpt of the Radio Frequency Warner System (the RFW product line) taken from [RFBRC+12] and written in OVM. This fictional system is used to assist drivers providing information about relevant traffic signs that are equipped with a radio-frequency identification tag (RFID).
2. Figure 4.1b shows an excerpt of the dopler model describing the variability of the Dopler tool suite taken from [MGH+11].

4.2 An Introduction to HLVL

This section informally presents the HLVL language focusing on the language's constructs and their semantics. Then, the following paragraphs illustrate the language using the running example and providing code snippets. The section closes presenting how HLVL can be used to represent variation-point-oriented languages and decision-oriented languages.

4.2.1 Models in HLVL

Variability models in HLVL are defined in terms of *options*, *domains*, and *variability relations*, the major concepts in the language. Models in HLVL have an identifier followed by two non-empty blocks containing the definitions

regarding variable items and their relations. The first block contains the options and domains for representing *what varies* in a particular system. The second block contains the variability relation as the rules to define *how options vary* given the rules in a particular domain.

The following example shows an extract of the parking assistant system product line containing two options and one relation. In the example, the options represent the optional inclusion of a processor and a memory and the relation represents implication between those options.

```
1 model Parking_Assistant_System
2 options:
3   choice memory, processor
4 relations:
5   r2: implies(processor, memory)
```

4.2.2 Options, Domains, and Variants

The variability units in HLVL are called *options*, they represent the variable items in a system that must be chosen or defined in a configuration process. For instance, the inclusion of a processor, the type of the screen, and the number of feedback devices are options in the running example. Each option is associated with a *domain*, representing the set of available choices an option can be bound to, in the configuration process. These available options are called *variants*. For example, the domain of the type of screen in the PAS product line has three variants, the set of available choices for the screen, that is, `monochromatic`, `color`, or `high-resolution`. The following sections present the domains and the types of options in HLVL.

Domains

HLVL supports boolean and non-boolean domains. Domain definitions contain the keyword `domain` and the set of variants described by extension using intervals or lists of values. Examples of domains defined as lists are the available sizes of memory `[2, 4, 8, 16, 32]`, and the set of types of screen `['basic', 'color', 'highRes']`. An example of domains defined as intervals is the number of cores in a processor defined as `1..7`. Section 4.3 presents the formal syntax with the rules to describe domains.

Options

HLVL supports three types of options to describe variability-intensive systems: *single-choice options*, *multiple-choice options*, and *attributes*.

Single-choice options are options bounded to exactly one variant in the domain during the configuration process. HLVL differentiates single-choice options considering the number of alternatives in the domain between *choices* and *enumerations*.

- **Choices** are single-choice options with exactly two alternatives. They are useful to represent Boolean variability units. Choices are special cases where there is no need to declare the alternatives in the domain because they will be encoded to boolean domains. In this way, choices can represent binary options such

as (1) to include/exclude an element from a product; (2) to answer yes/no, or true/false to a question or decision; and (3) to support/not support a particular characteristic in a configuration, among others. The keyword `choice` followed by an identifier declares a choice in a model.

Choices can be used for representing many variability units, for instance, features in feature-oriented models, variation points and variants in OVM models, and boolean decisions in decision oriented-models, among others. The following example shows how each feature in the PAS feature diagram is represented by a choice.

```
1  model Parking_Assistant_System
2  options:
3    choice memory, processor, sensor,
         speedSensor, positionSensor,
         feedback, audio, vibration,
         visual
```

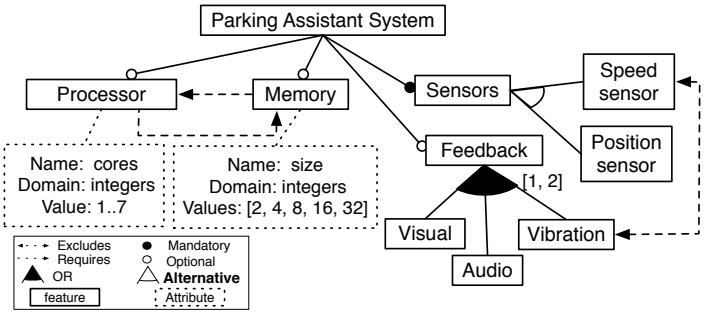


Figure 4.3: Defining a set of features as choices in HLVL

- **Enumerations** are single-choice options with more than two variants in the domain. They are useful to represent non-boolean variable items, such as, numeric features in feature-oriented models and decisions with a single choice in decision-oriented models. Enumerations are defined using the keyword `enum`, followed by one identifier and a domain. The following code snippet shows two `enum` options in the PAS product line.

```
1  enum screen domain: ['basic', 'color', 'highRes']
2  enum speakerType domain: 1..5 comment: {"What is the speaker type?"}
```

The first line in the example shows the variability of the type of screen represented as a non-numeric `enum` with alternatives `'basic'`, `'color'`, and `'high-resolution'`. In line two, the example shows the declaration of the variability in the type of speaker, represented as a numeric `enum` ranging in the interval 1..5. This example also shows that options may contain a comment to introduce annotations.

Multiple-choice options are options bounded to one or more variants in the domain during the configuration process. To declare multiple-choice options, HLVL provides the keyword `set`. Sets in this language can be used to represent decisions with more than one variant selected. The following example illustrates the inclusion of a multiple-choice option named `consistencyChecker` to model that some configurations of the PAS contain one or more consistency checker devices, one per type of sensor.

```
1  set  consistencyChecker domain: ['posChecker', 'speedChecker']
```

In the previous example, the option `consistencyChecker` can be bound in the configuration to variants`posChecker`, `speedChecker`, or both {`consistencyChecker, posChecker`}.

Attributes are particular elements in variability modeling languages. They represent properties or particular characteristics of variable items in a system. For instance, the size of the memory and the number of cores in a processor in the PAS. Many variability modeling languages include the concept of attributes, yet, there is no consensus on a notation to define attributes. Most proposals agree that their definition should include a type, a name, a domain, and optionally a *value* [BSRC10]. Differently than options, attributes can appear as operands in constraint expressions, then, it is important to include the syntactic elements to represent attributes. In HLVL, attribute declarations contain the keyword `att` followed by a type, a name, and a domain. The following example shows the declaration of `cores` and `size`.

```
1 // attributes
2 att integer cores domain: 1..7
3 att integer size domain: [2, 8, 16, 32]
```

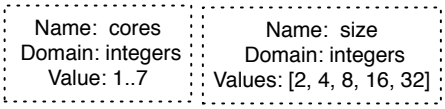


Figure 4.4: Attributes in HLVL

A note about Multiplicity. Options in HLVL can represent variable items that repeat themselves in a configuration, *i.e.,* have multiple instances with global semantics as described in [MCHB11]. Syntactically, an option may be declared with an interval to denote the allowable number of items in a configuration as shown in the following example:

```
1 choice processor [0,4]
2 enum screen [0,2]
3   domain: ['basic', 'color', 'highRes']
```

Let's suppose that the PAS may contain at most four processors and two screens. This example shows how this can be declared including the annotation `[min,max]` indicating the minimum and maximum number of variable items.

4.2.3 Variability Relations

Variability relations in HLVL are the rules to decide which variants, from the option's domain, should be selected in the configuration process. The language provides three types of variability relations:

- Rules for describing how the selection of a variant implies/excludes the selection of another variant or set of variants.
- Rules for describing hierarchies, parent-child relations involving single and multiple children.
- Rules to restrict the visibility of other variability relations.

The following subsections describe the different types of variability relations in the language.

Exclusion/Implication Relations.

Exclusion/Implication rules are useful for defining the system's variability in terms of which variants are compatible or require other specific variants, as well as which ones are not compatible at all. These rules can be described either using *keywords* or *constraint expressions*.

Constraint expressions are enough to define simple and complex exclusion/implication relations. However, the results in Chapter 3 showed that constraint expressions by themselves undermine the ontological clarity of the language. For example, the modeler will face confusion about the different options for representing a particular exclusion/inclusion relation. To ease this situation, the language provides the keywords to describe recurrent exclusion/exclusion relations. These keywords provide the modeler with constructs to represent common variability relations between Boolean options or choices as they are the variable units encoding features and variation points.

Keywords. HLVL provides three keywords to describe recurrent exclusion/exclusion relations on choices: **common**, **mutex**, and **implies**.

The keyword **common** is used to explicitly declare that a choice will be selected in all configurations. This construct can be used to define the root feature in feature-oriented models or mandatory variation points in variation-point-oriented languages. In HLVL, this is expressed as follows:

```
1 common(sensors, PAS)
```

The keyword **mutex** can be used to declare two types of conditional exclusion: *mutual exclusion* and *guarded exclusion*.

- Mutual exclusion is the relation where two choices cannot be part of the same configuration, similar to the mutual exclusion or *excludes* operator in feature-oriented and variation-point-oriented languages. In the running example, the mutual exclusion of the speed sensor and the position sensor is written as follows:

  ```
  1 mutex(speedSensor, positionSensor)
  ```

- Guarded exclusion is a relation containing a complex condition to exclude a group of choices. That is, whenever the condition is satisfied the group of choices wont be included in a configuration. For example, this type of relation can be used to define that sporadic clients cannot use the payment by gift card and debit card functionalities in HLVL as follows: ejemplo pendiente

The keyword **implies** can be used to represent three types of conditional inclusion: *implication*, *guarded implication*, and *quantified implication*.

- Implication is a binary relation among two choices to describe that the inclusion/selection of a choice implies the inclusion of another. This behavior shares the semantics of the *requires* operator in feature-oriented and variation-point-oriented languages. The following example shows how to define the implication relation between a processor and a memory device in the running example:

  ```
  1 implies(processor, memory)
  2 implies(memory, processor)
  ```

- Guarded implication is a relation stating a complex condition to be validated for the inclusion/activation of a group of choices. Then, whenever the condition is satisfied, the group of choices are present in the configurations.

- Quantified implication describe conditions regarding the number of instances of an element that are required to include a number of instances of another element. This type of relation, introduced by Quinton et al. [QRD13], is useful when the model includes the concept of multiple instances of the same choice. For example, let's include a made-up constraint in the problem: if there is at least two feedback sensors and at most four, then at most 2 processors are required. This is written in HLVL as follows:

```
1  imp3: [2, 4] feedback implies [1, 5] processor
```

Figure 4.5 gathers the examples of the inclusion/exclusion relations described above.

```
1  common(PAS)
2  mutex(speedSensor, positionSensor)
3  implies(processor, memory)
4  implies(memory, processor)
5  [2, 4] feedback implies [1, 5] processor
```

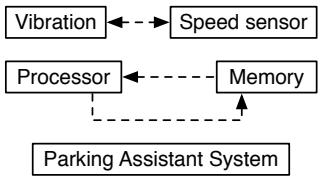


Figure 4.5: Examples of inclusion/exclusion relations in HLVL

Constraint Expressions. Constraint expressions in HLVL are useful for including complex rules between options in the variability model using logic, relational, arithmetic and global operators. Constraint expressions start with the keyword **`expression`** followed by an expression written in the HLVL's expression language summarized with the formal syntax of the language in Section 4.3. The expression language serves also to define the conditions in other variability relations, such as, conditional exclusion/inclusion, and visibility relations. Let's take a look to a constraint expression in the running example that can be written in different ways regarding the option's type (*e.g.,* Boolean, numeric, etc). Figure 4.6 shows the different ways to describe the mutual exclusion in the PAS system where the system does not use a speed sensor when it also contains a vibrating feedback device.

```
1  e1: expression(speedSensor >=1 => positionSensor = 0)
2  e2: expression(~( selected(speedSensor) AND selected(positionSensor)))
3  e3: expression(~selected(speedSensor) OR ~selected(speedSensor))
4  e4: expression(speedSensor + positionSensor <= 1)
```

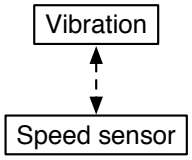


Figure 4.6: Elements and variants in HLVL

Hierarchy and Parent-Child Relationships

Although HLVL is not a language where hierarchical relations are essential for composing models, it offers a set of constructs to describe one-to-one (parent-child), and one-to-many (parent-children) relations.

One-to-one relations are represented with the keyword **`decomposition`** followed by the names of the parent and child options together with a multiplicity (*i.e.,* , multiplicities are also called cardinalities [BSRC10]). This

multiplicity is an annotation in the form $[min, max]$ used to bound the number of instances on each child option. The following are three forms to use one-to-one parent-child relationships to describe particular situations in variability modeling.

1. Decompositions with multiplicities $[1, 1]$ and $[0, 1]$ are special cases to represent the constructs *mandatory* and *optional*, respectively [CHE05]. Mandatory and optional are basic constructs present in most feature-oriented and variation-point-oriented languages. In the running example, the relations stating that the inclusion of processor, feedback, and memory are optional and the inclusion of sensors is mandatory are written in HLVL as shown in Figure 4.7.

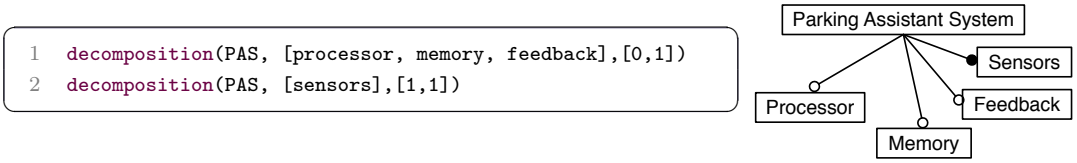


Figure 4.7: Examples of hierarchy *one-to-one* relations with $[0, 1]$ and $[1, 1]$ multiplicities

2. Decompositions with multiplicity $[1, 1]$ are also used to associate options to its attributes. Our running example contains two attributes `cores`, and `size` declared as decompositions in Figure 4.8. The inclusion of these decompositions enable the qualified names `processors.cores`, and `memory.size` to be used as identifiers in constraint expressions or conditions.

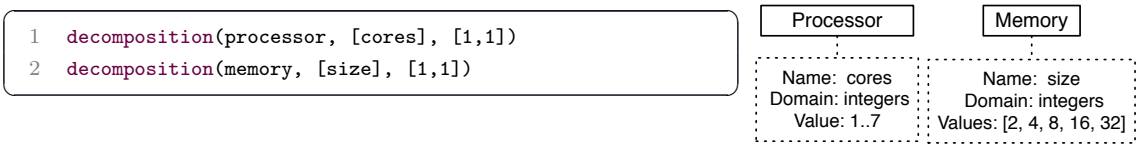


Figure 4.8: Examples of hierarchy *one-to-one* relations to link options and attributes

3. Decomposition with multiplicities in the form $[m, n]$ where $n > 1$ can be used to represent options that have multiple instances with local semantics as described in [MCHB11]. For example, let's include in the example some rules regarding the number of position sensors and speed sensors. Figure 4.9 shows how to use cardinalities to include two multiplicity declarations for defining that the PAS supports between zero and three speed sensors and zero and two position sensors.

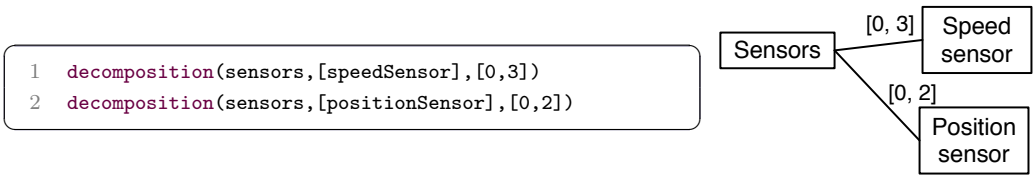


Figure 4.9: Examples of hierarchy *one-to-one* relations to declare multiplicities with local semantics

One-to-many hierarchical relations are represented with the keyword `group`. These relations contain the identifier of the parent option, the children option's identifiers, and a multiplicity, also called group cardinality. This multiplicity is used to specify the minimum and the maximum number of children in the group that can appear in a configuration. The language provides the symbol `*` to be included in multiplicities in the form `[1,*]` to denote that at least one, and at most the total number of children can be selected/included in the configuration. Figure 4.10 shows how to model a group relation between the feedback devices in the running example.

```
1  group(feedback, [audio, vibration, visual],[1,2])
```

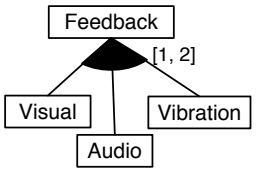


Figure 4.10: Example of hierarchies one-to-many groups in HLVL

Visibility

Visibility relations in HLVL are rules to condition the availability (*i.e.,* hide) of a group of options and their relations with similar semantics than visibility rules in decision models [DHR10]. These relations are declared starting with the keyword `visibility` followed by a constraint expression and the identifiers of the options this condition hides. For example, let's imagine that a controller device used to check the consistency of the sensors and feedback devices should be included when the number of sensors and feedback devices is greater than four. Figure 4.11 shows how to describe a visibility relation in HLVL. This example does not have a graphical depiction since FODA-related languages do not include constructs to model visibility.

```
1  visibility(( instances(positionSensor) +
2                           instances(speedSensor) +
3                           instances(visual) +
4                           instances(audio) +
5                           instances(vibration)) >= 4, [
                                consistencyChecker])
```

Figure 4.11: Declaring visibility relations in HLVL

4.2.4 What about other variability Languages?

The previous section introduced the language and its constructs using an example originally presented as a feature model in the extended notation of Benavides *et al.* [BTRC05a]. This example served to demonstrate the HLVL can be used for modeling feature-oriented models. The following subsections show, through examples, how specific constructs of variation-point-oriented and decision-oriented languages can be represented as well with the HLVL constructs described before.

Modeling Variation-Point Languages in HLVL

To illustrate how to encode variation-point oriented languages in HLVL, recall the Radio Frequency Warner system (RFW) product line written in OVM and presented at the begging of this chapter in Figure 4.1a and depicted again in Figure 4.12 to ease the reading. This figure depicts an excerpt of the Radio Frequency Warner System (the RFW product line) taken from [RFBRC+12] and written in OVM. This fictional system is used to assist drivers providing information about relevant traffic signs that are equipped with a radio-frequency identification tag (RFID).

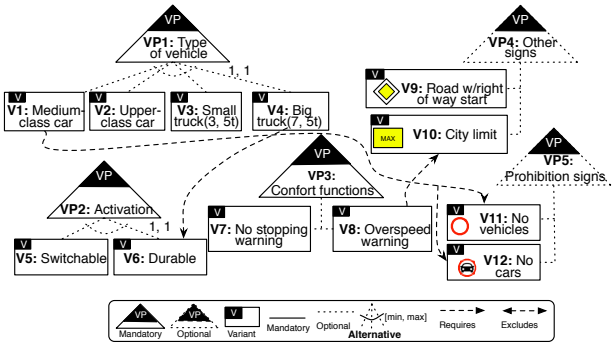


Figure 4.12: Orthogonal Variability Model (OVM) for the RFW product line. Adapted from [RFBRC+12]

Variation Points (VP) and variants can be modeled using different types of options in HLVL, such as, choices and enumerations. For example, the variation point VP5 and its variants V11 and V12 in the running example can be modeled as choices as shown in Figure 4.13. Variation points and variants linked with alternative [1..1] are special cases that can be modeled using both choices and enumerations. Line 4 in the example shows how to model VP2 as an enumeration with the list of variant's ids as domain. This encoding works because in the configuration process enumerations are bounded to only one value in their domain.

```
1  choice VP5 comment:{"Prohibition signs"}
2  choice V11 comment:{"No vehicles"}
3  choice V12 comment:{"No cars"}
4  enum VP2 domain: ['V5', 'V6'] comment:{"Activation"}
```

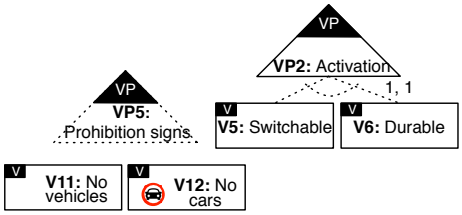


Figure 4.13: Representing variation points and variants with choices and enumerations in HLVL

Figure 4.13 shows how each variation point and variant has an identifier and a description for extra information. These descriptions, were included using the keyword `comment` in the definition of the element that allows the modeler to label or comment the delaration.

Mandatory variation points are items that are always bounded in a configuration. The construct `common` explicitly defines that a set of elements must be bounded in the configuration. In the example, `VP1`, `VP2`, and `VP3` are mandatory, this is expressed in HLVL as shown in Figure 4.14.

The links between variation points and its variants (*i.e.,* mandatory, optional, and alternative) represent one-to-one and one-to-many hierarchical relations. On the one hand, mandatory and optional links can be

```
1  common(VP1, VP2, VP3)
```

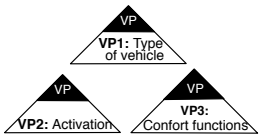


Figure 4.14: Defining mandatory variation points in HLVL

represented in HLVL using the **decomposition** construct. On the other hand, alternative $[m, n]$ links can be represented using the **group** construct. For instance, Figure 4.15 shows the optional relation between VP5 and V11, V12, and the alternative relation between VP1 and V1, V2, V3, V4.

```
1  decomposition(VP5, [V11, V12], [0,1])
2  group(VP1, [V1, V2, V3, V4], [1,1])
```

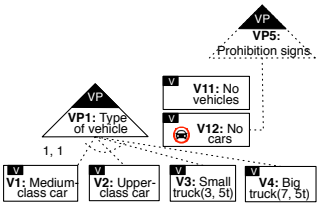


Figure 4.15: Modeling OVM links in HLVL

Constraints in OVM models can be represented with the constructs **expression**, **implies**, and **mutex**. Line 1 in Figure 4.16 shows the representation of the implication relationship between V8 and V10. More complex constraints are required when the modeler choose to represent alternative [1..1] relations using enumeration. In this case, the constraints in the OVM model can be represented with constraint expressions and guarded implications as shown in the example depicted in Figure 4.16.

```
1  implies(V8, V10)
2  expression(VP1 ='big truck' => VP2 ='durable')
3  implies(VP1 ='medium-class car', [V11,V12])
```

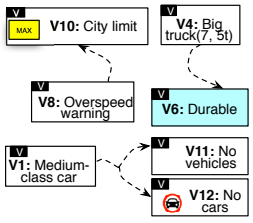


Figure 4.16: Constraint expressions in HLVL

HLVL can also encode other variation-point-oriented languages and OVM extensions. For instance, the attributes in Roos *et al.* [RFBC10, RFBRC+12] can be included using choices, attributes and decompositions. Also, the special constraints included in the proposal of Dumiterscu *et al.* [DTS+14] are supported by HLVL's expression language.

Modeling Decision Models in HLVL

The following paragraphs describe how to encode decision-oriented languages using HLVL. This description will be illustrated using the dopler model describing the variability of the Dopler tool-suite presented in Section 4.1.1 and depicted for better readability in Figure 4.17.

Decision-oriented languages have two variability units called decisions and assets. Decisions are variable items containing many pieces of information. Decisions contain types, cardinalities, and attributes. Decision types range between Boolean, String, Number, and Enumeration. The cardinality in a decision defines the minimum and maximum number of values to be selected. The attributes in a decision contain annotations, useful to present the questions that guide the configuration process. Consider the decision highlighted in blue in Figure 4.17, it is a decision of type enumeration with cardinality 1:1 and the annotation *What to buy?*.

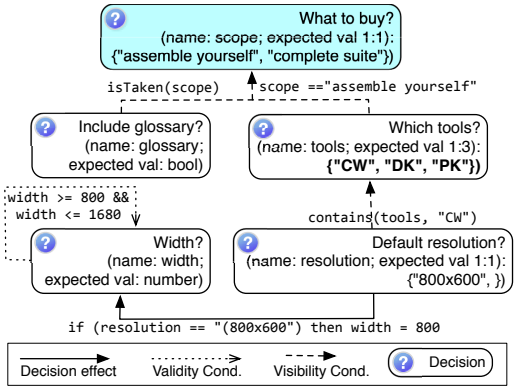


Figure 4.17: Dopler Model for the Dopler tool-suite taken from [MGH+11]

Decisions with cardinality $1:1$ can be modeled using choices or enumerations. Boolean decisions are encoded to choices and the other type are encoded to enumerations. Figure 4.18 shows the representation of two $1:1$ decision in the running example: `scope` and `glossary`. Decisions with cardinality $1:N$ can be represented using sets in HLVL. For example, line 3 in Figure 4.18 shows how the decision `tools`, its three variants, and its cardinality are represented in HLVL.

```
1 enumeration scope domain: ['assemble yourself', 'complete suite'
      ] comment: "What to buy?"
2 choice glossary comment:"Include glossary?"
3 set tools domain: ['confWizard','decisionKing','projectKing']
      comment:"Which tools?"
```

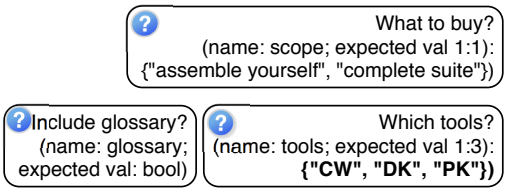


Figure 4.18: Constraint expressions in HLVL

Figure 4.19 depicts three examples of variability relations in Dopler models: visibility conditions, decision effects, and validity conditions. Visibility conditions in decision models are modeled in HLVL using the `visibility` construct. In the example, we use the visibility construct to describe that the decision about the resolution becomes visible when the user decides to include the configuration wizard tool. Decision effects in dopler models describe dependencies between decisions as rules triggering values for other decisions. The second line in the example shows the constraint expressions representing the rule determining that the selection of the resolution triggers the value of the width. Validity conditions are the rules restricting the range of the values which can be assigned to a decision. In HLVL, these rules are written using constraint expressions. For example, Figure 4.19, line 3 shows the validity condition restricting the width as a number between [800, 1680].

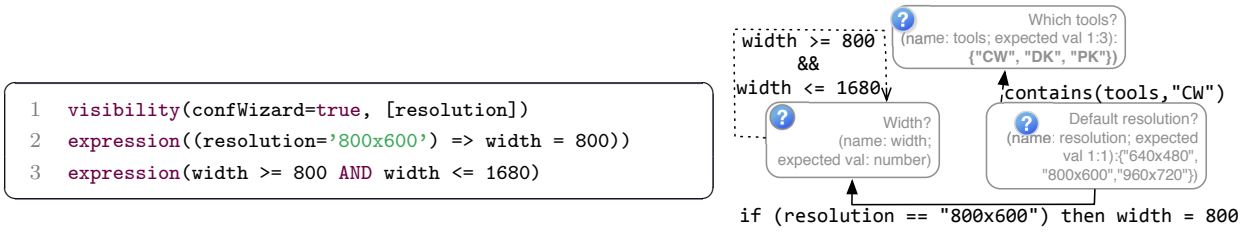


Figure 4.19: Constraint expressions in HLVL

4.3 Formal Syntax

This section presents the formal syntax using a context free grammar in the extended Backus-Naur Form (EBNF) as follows:

- Production rules have the form $left = right$; where $left$ is a non-terminal, right is a combination of non-terminals, terminals, and symbols. The semicolon is the termination symbol.
- Non-terminals are enclosed in angular brackets as in $\langle option \rangle$. Non-terminals are named using the Camel case naming convention[1].
- Terminal symbols are enclosed in double quotes. Terminal symbols colored in purple correspond to keywords.
- The symbols ?, $*$, and $+$ represent shortcuts. $S?$ means that S is optional, S^+ means that S appears at least once, and S^* means that S appears zero or many times.
- The rules for declaring options use the non-terminal $\langle name \rangle$ representing identifiers. Section 4.3.5 describes the well-formedness rules for defining identifiers, literals, and other syntactic categories.
- The rules for declaring variability relations use *metavariables* such as E_i, C_i, S_i representing names for different types of options and R_i for representing relation's names.

Table 4.1 presents a summary of the most important production rules in the language. The following subsections describe the production rules and the syntactic categories in the table.

4.3.1 Rules for Models

The starting non-terminal is $\langle model \rangle$. A model in HLVL has an identifier followed by two non-empty blocks containing options and relations as shown in production rule 4.1.

$$\langle model \rangle = \texttt{model}\ \langle name \rangle\ \texttt{options:}\ \langle options \rangle^+\ \texttt{relations:}\ \langle relation \rangle^+ \tag{4.1}$$

[1]Camel case is the practice of writing phrases without spaces or punctuation, indicating the separation of words with a single capitalized letter

Table 4.1: Formal syntax of HLVL in EBNF

$\langle model \rangle =$	`model` $\langle name \rangle$ `options:` $\langle options \rangle^+$ `relations:` $\langle relation \rangle^+$	4.1
$\langle options \rangle =$	`choice` $\langle name \rangle$ $\langle multiplicity \rangle?$ $\langle comment \rangle?$ |	4.2
	`enum` $\langle name \rangle$ $\langle multiplicity \rangle?$ `domain:` $\langle domain \rangle$ $\langle comment \rangle?$ |	4.3
	`set` $\langle name \rangle$ $\langle multiplicity \rangle?$ `domain:` $\langle domain \rangle$ $\langle comment \rangle?$ |	4.4
	`att` $\langle type \rangle \langle name \rangle$`domain:` $\langle domain \rangle$ $\langle comment \rangle?$ |	4.5
	`att`$\langle type \rangle \langle name \rangle$`is` $\langle literal \rangle$ $\langle comment \rangle?$	4.6
$\langle domain \rangle =$	$\langle numericLiteral \rangle$`".."`$\langle numericLiteral \rangle$ |	4.7
	`"["` $\langle literal \rangle$`( ","` $\langle literal \rangle)^*$ `"]"`	4.8
$\langle relation \rangle =$	$\langle name \rangle$ `":"` $\langle variabilityRelation \rangle$	4.9
$\langle variabilityRelation \rangle =$	`common"("` C_1`( ","` $C_k)^*$ `")"`|	4.10
	`mutex "("` C_1 `","` C_2 `")"` |	4.11
	`mutex"("` $\langle constraintExpression \rangle$ `","` C_1`( ","` $C_k)^*$ `")"` |	4.12
	`implies "("` C_1 `","` C_2 `")"` |	4.13
	`implies"("` $\langle constraintExpression \rangle$ `","` C_1`( ","` $C_k)^*$ `")"` |	4.14
	$\langle multiplicity \rangle$ C_1 `implies` $\langle multiplicity \rangle$ C_2 |	4.15
	`expression "("` $\langle constraintExpression \rangle$ `")"`|	4.16
	`decomposition"("`P `","` `"["` C_1`(","` $C_k)^*$ `"]"` `","` $\langle multiplicity \rangle$ |	4.17
	`group"("`P `","` `"["` C_1`(","`$C_k)^*$ `"]"` `","` $\langle multiplicity \rangle$ |	4.18
	`visibility "("` $\langle constraintExpression \rangle$ `","` `"["` C_1`(","`$C_k)^*$ `"]"`	4.19

The first block is the *options* block that contains the keyword `options:` and at least one variable item. The second block is the *variability relations* block containing the keyword `relations:` and at least one variability relation.

4.3.2 Rules for Options and Domains

There exists five rules to declare options in HLVL. Options can be defined either as a choice, enumeration, set, or attribute. In the case of attributes, they can be associated to a domain, or a single value. Rules 4.2 to 4.6 show the productions to define options.

$$\langle options \rangle = \texttt{choice}\ \langle name \rangle\ \langle multiplicity \rangle?\ \langle comment \rangle?\ | \tag{4.2}$$

$$\texttt{enum}\ \langle name \rangle\ \langle multiplicity \rangle?\ \texttt{domain:}\ \langle domain \rangle\ \langle comment \rangle?\ | \tag{4.3}$$

$$\texttt{set}\ \langle name \rangle\ \langle multiplicity \rangle?\ \texttt{domain:}\ \langle domain \rangle\ \langle comment \rangle?\ | \tag{4.4}$$

$$\texttt{att}\ \langle type \rangle \langle name \rangle \texttt{domain:}\ \langle domain \rangle\ \langle comment \rangle?\ | \tag{4.5}$$

$$\texttt{att} \langle type \rangle \langle name \rangle \texttt{is}\ \langle literal \rangle\ \langle comment \rangle? \tag{4.6}$$

Each option's declaration starts with a keyword to differentiate its type, followed by the option's unique name, and a domain. Non-terminals $\langle multiplicity \rangle$ and $\langle comment \rangle$ are optional and they are defined in Section 4.3.5 along with other syntactic categories such as names, literals, and types.

Rules 4.5 and 4.6 are used to define attributes. Each attribute is associated to a type, a name and a domain. HLVL supports boolean, integer and symbolic data types. Also, it is possible to associate an attribute to only one value with the construct `is` as shown in rule 4.6.

The following rules present the two different ways to declare domains, as intervals and as lists.

$$\langle domain \rangle = \langle numericLiteral \rangle \texttt{".."} \langle numericLiteral \rangle\ | \tag{4.7}$$

$$\texttt{"["} \langle literal \rangle (\texttt{","} \langle literal \rangle)^* \texttt{"]"} \tag{4.8}$$

On the one hand, intervals can be used only to define domains containing numeric literals (*cf.* Rule 4.7). On the other hand, lists may contain symbolic or numeric literals with the restriction that all literals belong to the same type (*cf.* Rule 4.8). As described in Section 4.2 there is no need to declare domains for choices as they always represent Boolean variants.

4.3.3 Rules for Variability Relations

Variability relations are named. These names are useful for referencing purposes. Then, each variability relation contains an identifier and a variability relation.

$$\langle relation \rangle = \langle name \rangle \texttt{":"} \langle variabilityRelation \rangle \tag{4.9}$$

In general, variability relations contain a keyword and a set of parameters enclosed by parenthesis and/or brackets. HLVL provides ten constructs to declare variability expressions. The following production rules describe how to derive variability relations in HLVL. Note that, these rules use metavariables such as C_i, P_i, and C_i for

naming elements and R_i for naming variability relations.

$$\langle variabilityRelation \rangle = \texttt{common "("}\ C_1\ \texttt{(","}\ C_k)^*\ \texttt{")"} \mid \tag{4.10}$$

$$\texttt{mutex"("}\ C_1\ \texttt{","}\ C_2\ \texttt{")"} \mid \tag{4.11}$$

$$\texttt{mutex"("}\ \langle constraintExpression \rangle\ \texttt{","}\ C_1\ \texttt{(","}C_k)^*\ \texttt{")"} \mid \tag{4.12}$$

$$\texttt{implies"("}\ C_1\ \texttt{","}\ C_2\ \texttt{")"} \mid \tag{4.13}$$

$$\texttt{implies"("}\ \langle constraintExpression \rangle\ \texttt{","}\ C_1\ \texttt{(","}C_k)^*\ \texttt{")"} \mid \tag{4.14}$$

$$\langle multiplicity \rangle\ C_1\ \texttt{implies}\ \langle multiplicity \rangle\ C_2 \mid \tag{4.15}$$

$$\texttt{expression "("}\ \langle constraintExpression \rangle\ \texttt{")"} \mid \tag{4.16}$$

$$\texttt{decomposition"("}P\ \texttt{","}\ \texttt{"["}\ C_1\ \texttt{(","}\ C_k)^*\ \texttt{"]"}\ \texttt{","}\ \langle multiplicity \rangle \mid \tag{4.17}$$

$$\texttt{group"("}P\ \texttt{","}\ \texttt{"["}\ C_1\ \texttt{(","}\ C_k)^*\ \texttt{"]"}\ \texttt{","}\ \langle multiplicity \rangle \mid \tag{4.18}$$

$$\texttt{visibility"("}\ \langle constraintExpression \rangle\ \texttt{","}\ \texttt{"["}\ C_1\ \texttt{(","}\ C_k)^*\ \texttt{"]"} \tag{4.19}$$

The `common` construct, defined in Rule 4.10, declares a collection of choices (O_i) separated by a comma. The choices under this relation will appear in all configurations.

Rules 4.11 and 4.12 present the two forms to declare a mutual exclusion. The first rule declares the mutual exclusion of two choices, while the second contains a constraint expression that conditions exclusion of the set of elements choices in brackets.

Rules 4.13, 4.14, and 4.15 define the three ways to declare implication relations among choices. Rule 4.13 presents the simple implication which applies to a pair of choices. Rule 4.14 defines the guarded implication. This relation conditions the inclusion of a set of choices separated by a comma and enclosed in brackets. Rule 4.15 defines the quantified implication. Quantified implications start with a pair multiplicity-choice, followed by the keyword `implies`, and a second pair multiplicity-choice to close. The quantified implication appears in the model when it contains options with multiple instances.

Rule 4.16 presents the `expression` construct. Constraint expressions define complex rules among options. These expressions are written in the HLVL's expressions language (*cf.* Section 4.3.4).

Rule 4.17 defines the syntax of the `decomposition` construct. Decomposition relations contain a parent option, a set of children options enclosed in brackets, and a multiplicity annotation. Multiplicities are defined according to Rule 4.35 described with the other well-formedness rules in Section 4.3.5.

Rule 4.18 defines the syntax of the `group` construct. A group relation contains the keyword `group` followed by the identifier of one parent option, a set of children's identifiers enclosed in brackets, and a multiplicity. This multiplicity specifies the minimum and the maximum number of children in a configuration when the parent is included.

Rule 4.19 presents the `visibility` construct. A visibility relation contains a constraint expression and a set of identifiers enclosed in brackets.

4.3.4 Expressions Language

HLVL provides an expressions language for representing constraint expressions in HLVL. A *constraint expression* is an infix expression built using values, names and operators. Constraint expressions are always evaluated to boolean values. The following production rules define the expressions language.

$$\langle expression \rangle = \sim \langle boolExp \rangle \mid \langle boolExp \rangle \mid \langle relational \rangle \tag{4.20}$$

$$\langle boolExp \rangle = \langle booleanLiteral \rangle \mid \langle name \rangle \mid \langle boolExp \rangle \langle logicOp \rangle \langle boolExp \rangle \mid \tag{4.21}$$

$$\langle relational \rangle = \langle arithmetic \rangle \langle relationalOp \rangle \langle arithmetic \rangle \tag{4.22}$$

$$\langle arithmetic \rangle = \langle name \rangle \mid \langle numericLiteral \rangle \mid \langle arithmetic \rangle \langle arithmeticOp \rangle \langle arithmetic \rangle \mid \langle unaryFunction \rangle \texttt{"("} \langle arithmetic \rangle \texttt{")"} \mid \langle binaryFunction \rangle \texttt{"("} \langle arithmetic \rangle \texttt{","} \langle arithmetic \rangle \texttt{")"} \tag{4.23}$$

$$\langle logicOp \rangle = \texttt{AND} \mid \texttt{OR} \mid \texttt{=>} \mid \texttt{<=>} \tag{4.24}$$

$$\langle RelationalOp \rangle = \texttt{=} \mid \texttt{!=} \mid \texttt{>} \mid \texttt{>=} \mid \texttt{<} \mid \texttt{<=} \tag{4.25}$$

$$\langle arithmeticOp \rangle = \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{mod} \tag{4.26}$$

$$\langle unaryFunction \rangle = \texttt{abs} \mid \texttt{sqrt} \tag{4.27}$$

$$\langle binaryFunction \rangle = \texttt{pow} \mid \texttt{min} \mid \texttt{max} \tag{4.28}$$

Constraint expressions defined in Rule 4.20 are a boolean expressions or its negation. Boolean expressions can be composed using logic operators or relational expressions. Rules 4.21 to 4.28 describe the syntax of boolean expressions using logic operations, relational expressions, and basic arithmetic expressions.

4.3.5 Well-Formedness Rules

This section presents a set of rules a modeler has to adhere in order to produce valid scripts and cannot be represented using a free-context grammar.

Naming Rules.

The rules to naming the options and variability relationships are similar to the rules in programming languages. First, names can have letters, digits, underscore and cannot start with digits, special characters, contain spaces or use keywords.

$$\langle name \rangle = \langle ID \rangle \mid \langle ID \rangle.\langle ID \rangle \tag{4.29}$$

$$\langle ID \rangle = (\texttt{"a"..."z"} \mid \texttt{"A"..."Z"})(\texttt{"a"..."z"} \mid \texttt{"A"..."Z"} \mid \texttt{"\_"} \mid \texttt{"0"..."9"})^*$$

Names have to be unique within their scope. The following rules prevents a number of potential naming conflicts

- The model, options and variability relations have unique names.
- Enumeration values in `enum` options. Furthermore, no enumeration values may have the name of the model, options or attributes.
- Names should be declared before their use.
- HLVL is a case sensitive language.

Qualified Names. The link between options and attributes enable the use of qualified names. Qualified names can be used inside a constraint expression, they start with the name of the option followed by a dot and the name of the attribute as described in Rule 4.29

Literals and Types

The Literals are the values that can be used in expressions or as variants in a domain. The HLVL supports Boolean literals, numeric literals and symbolic literals. The following are the rules to define literals in the language.

$$\langle literal \rangle = \langle booleanLiteral \rangle \mid \langle numericLiteral \rangle \mid \langle symbolicLiteral \rangle \tag{4.30}$$

$$\langle booleanLiteral \rangle = \texttt{"true"} \mid \texttt{"false"} \tag{4.31}$$

$$\langle numericLiteral \rangle = \langle integer \rangle \mid \langle real \rangle \tag{4.32}$$

$$\langle symbolicLiteral \rangle = \texttt{"'"}\langle string \rangle\texttt{"'"} \tag{4.33}$$

HLVL is strongly typed, and does not support type casting. Type correctness is defined as follows.

- Constraint expressions should be evaluated as Boolean values.
- Variants in a domain are literals of the same type.
- Expressions involving enumerations may only use values defined in the domain of the enumeration.
- The types associated to attributes are as follows:

$$\langle type \rangle = \texttt{"boolean"} \mid \texttt{"integer"} \mid \texttt{"real"} \mid \texttt{"symbol"} \tag{4.34}$$

Other Rules

Multiplicities in the for $[m, n]$, $m <= n$ are present when declaring options, quantified implication, decompositions, and groups. They denote the minimum and maximum number of elements that could be number of instances or number of children in configurations. Multiplicities are defined as follows.

$$\langle multiplicity \rangle = \texttt{"["}\langle integer \rangle\texttt{","}\langle integer \rangle\texttt{"]"} \tag{4.35}$$

Comments are annotations optionally included when declaring options. These annotations provide extra information about the option and are particular useful to represent decisions in decision-oriented languages. The following rule defines comments.

$$\langle comment \rangle = \texttt{"comment:"}\ \langle string \rangle \tag{4.36}$$

4.4 Formal Semantics

The language semantics defines the meaning of a program, an expression, a model, or anything composed using the language's syntax. This meaning is a social construct known and well-understood by the target community [HR04]. In the variability modeling domain, the semantics of variability models are the set of valid configurations represented by the model [SHTB07, BHST04, DHR10]. These semantics, known as the configuration semantics, are usually described following the Harel & Rumple guidelines, as in the works of Schobbens *et al.* [SHTB07, BHST04], Dhungana et al. [DHR10], and Berger *et al.* [Ber12], among others.

There is another approach for defining the language semantics. This approach defines the semantics as the set of transformation rules to map variability models into constraint problems, such as SAT-problems, SMT problems, and constraint satisfaction problems, among others [BSRC10]. These transformation rules are called the operational semantics. The operational semantics of a language were first introduced by Gordon Plotkin in [Plo81] as a way to formalize the behavior of the language's constructs [SR90]. This formalization is desirable because it provides a clear and unambiguous language, independent of any implementation. Moreover, this formalization can be used to structure the language's implementation and answer questions about the interaction of a language such as, What does the compiler do? and How is the source code executed/evaluated/processed to obtain an answer?

The design of the HLVL's semantics considered a mixture of both approaches. Then, the remaining of this section presents (1) the concept-driven approach that guided the design of the operational semantics; (2) the elements connecting the operational semantics and the configuration semantics (*i.e.,* semantic domain and semantic function); and (3) the definition of the inference rules in the operational semantics of HLVL.

4.4.1 The HLVL(x) Sublanguages

The following subsections present the concept of sublanguage in the context of this thesis and the approach to define such sublanguages.

A Concept-Driven Approach

The idea of studying a language in terms of its underlying concepts was introduced by Abelson & Sussman [AS96] and later developed by Van Roy & Haridi [VRH04]. This idea consists of studying programming paradigms by introducing simple concepts and incrementally explain more sophisticated ones. Furthermore, Van Roy & Haridi organized the concepts into simple languages called *kernel languages*. Then, they map programming languages and paradigms to the closely related kernel languages to teach programming from a concept point of view instead of from a language point of view.

The design of HLVL shares Abelson & Sussman's view to study languages in terms of their underlying concepts. Consequently, HLVL's design followed an ontological approach aiming to (1) conceptualize and structure knowledge about variability modeling concepts; (2) study the commonality and variability among modeling languages; and (3) define a set of constructs that comprehensively represent variability concepts and modeling paradigms. Also, the design of HLVL's operational semantics follows Van Roy & Haridi's idea to assemble sublanguages by gathering sets of similar concepts to handle the expressiveness/analysis trade-off.

Van Roy & Haridi's map programming languages to one or more kernel languages to study programming paradigms in terms of its underlying concepts instead of the particularities of a language. The definition of the HLVL(x) sublanguages is consistent with this proposal. Particular variability modeling languages are mapped to one or more HLVL(x) sublanguages they are compatible with. This, mapping is useful to understand the expressiveness of the modeling language, its analysis requirements, and the definition and implementation of parsers. This idea will be developed in Chapter 5 in the description of the implementation of the analysis framework prototype.

Sublanguages and Support Levels

An HLVL(x) sublanguage is a set of constructs defining an expressiveness capability. These sublanguages are organized in a layered fashion, starting for a simple base language and successively adding new constructs. Sublanguages have a distinctive name inside the parenthesis, that is the reason why the complete set of sublanguages is called HLVL(x).

A **support level** indicates the type of solver required to support automated analysis for models in a particular sublanguage. Figure 4.21 summarizes how the sublanguages and support levels are organized according to this approach. Each container corresponds to a support level and solid boxes represent sublanguages. The inner box contains a simple variability language and successive boxes add concepts incrementally. Also, Figure 4.21 shows that a support level may contain more than one sublanguage.

The decision regarding which constructs to gather in a particular sublanguage considered (1) how these constructs support the concepts in the variability glossary; and (2) how they impacted the needs/limitations for automated analysis. The support levels and sublanguages are described as follows.

Level 1. Boolean Support contains the sublanguage whose models are represented as constraint problems with Boolean variables and Boolean constraints. This level supports the first sublanguage.

HLVL(*bool*) is the most basic variability language with choices for representing boolean variable items, hence its name. This sublanguage supports commonality, implication, mutex, parent-children relationships with limited multiplicities and constraint expressions over booleans and logic operators. This sublanguage is compatible with Basic FODA and OVM models and tools, such as, S.P.L.O.T [MBC09] and Batory's Tree Grammars [Bat05].

Level 2. Integer Support contains a set of sublanguages requiring support for integer variables, constraints, arithmetic and relational operations. The following are the sublanguages supported by this level.

HLVL(*att*) adds attributes and extends the expression language with numeric literals, arithmetic and relational operators. The inclusion of attributes has two important consequences. First, attributes enable

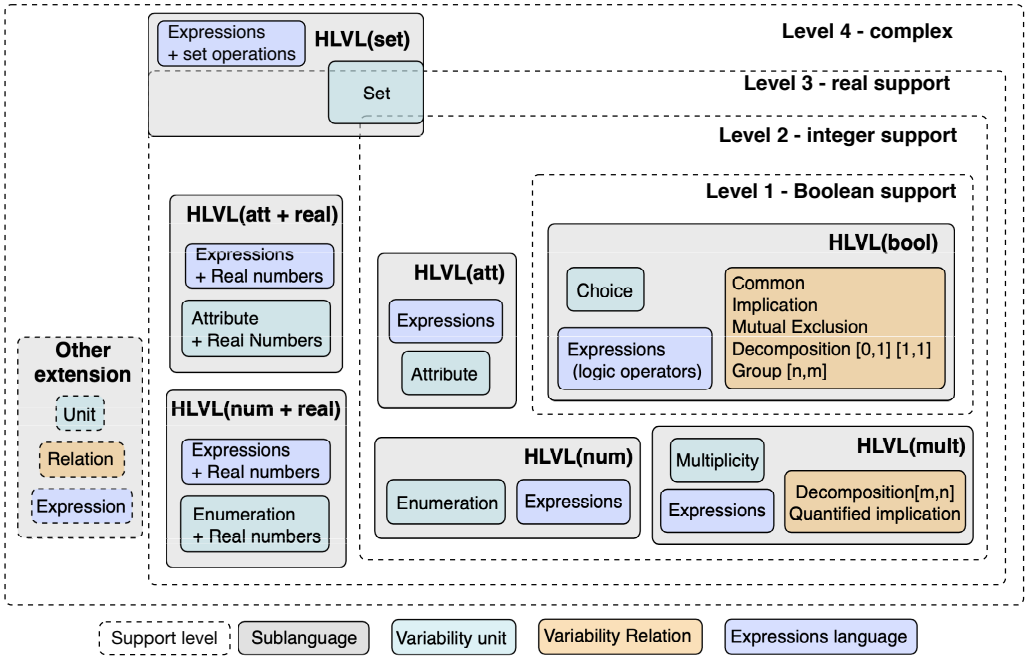


Figure 4.20: HLVL(x) Sublanguages

the usage of qualified names. Second, attributed models require more complex logic representations. This sublanguage is compatible with other attributted-based languages, *e.g.,* attributed-based feature models, attributed-based variation-point-oriented models [RFBC10], and tools like FeatureIDE.

HLVL(num) extends the basic language with enumerations and expressions with numeric literals, arithmetic and relational operators, excluding real numbers. This sublanguage supports decision-oriented languages (*e.g.,* DoplerML [DGR11]), feature-oriented languages with numeric features [MOP+19], and other languages with complex data types, such as, IVML [SKES18], and TVL [CBH11].

HLVL($multi$) includes the constructs for options with multiple instances in a configuration.

Level 3. Real-numbers Support is the level supporting languages with numeric variables and arithmetic expressions with float numbers. This level supports the following languages.

HLVL($att + real$) includes the same constructs than HLVL(att) plus real numbers support.

HLVL($num + real$) include the same constructs than HLVL(num) plus real numbers support.

Level 4. Complex CCP Support is the level that gathers languages requiring solvers supporting particular constraint systems, *e.g.,* Sets, Records, and Tuples, and extended constraint satisfaction problems, such as annotations, and logic quantifiers. This space may contain sublanguages in the future HLVL extensions.

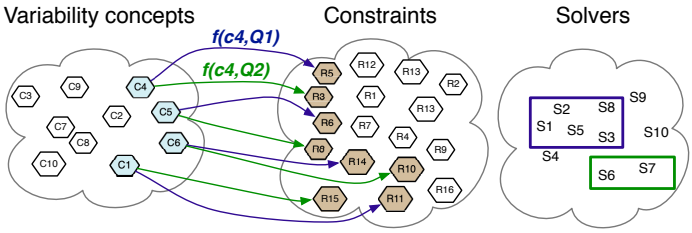
HLVL(set) is the language allowing the usage of `sets` and expressions with operations over sets. Sets are multivaluated options, that is, options that can have more than one value in the configuration. Though sets are partially supported by previous levels, sets operations are supported only with solvers supporting constraint systems over sets.

Orthogonal Constructs

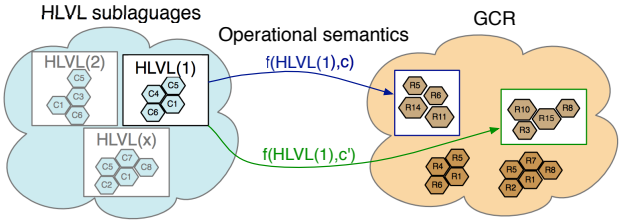
No sublanguage contains the constructs `before`, `visibility`, and the conditional `implies`, and conditional `mutex`, since those constructs do not have an effect on the expressiveness of the sublanguages. Then, those language constructs are somewhat orthogonal to all sublanguages and may be combined with any of them [TSS19].

4.4.2 Operational Semantics

The study of variability languages in terms of their concepts and the organization of those concepts into sublanguages and support levels is the preamble to define HLVL's operational semantics. Figure 4.21 illustrates the idea underlying the definition of HLVL's operational semantics taking into consideration the HLVL(x) sublanguages.



(a) Reasoning using sub-languages in Coffee



(b) Reasoning using sub-languages in Coffee

Figure 4.21: Variability models using different languages.

The operational semantics in a variability language works as a function $f : Construct, Context \times Constraints$, where a language construct C_i is encoded in a constraint regarding the context Q of the transformation. This context is defined at design time considering the logic paradigm and solvers supported by a particular tool. Figure 4.21a illustrates how the transformation function applied to the same concept may produce different constraints that are supported by solvers with different characteristics.

For example, consider the concepts in FODA models and two different tool-contexts, *i.e.,* S.P.L.O.T, and the stand-alone version of VariaMos tool-suite. In Figure 4.21a the concept C_4 (*e.g.,* mutual exclusion) is transformed in two different constraints R_5, R_3 (*e.g.,* $\neg(A \wedge B)$, $A + B \leq 1$) because the context Q_1 of S.P.L.O.T uses SAT solvers and CNF representations, while context Q_2, the VariaMos tool-suite, relies on CLP Prolog solvers and HLCL representations.

Figure 4.21b illustrates the idea in the approach developed in this thesis. This approach exploits the HLVL(x) sublanguages. The transformation function examines the context of the model and the context of the tool and

decides which transformation rules should be applied to procure a constraint satisfaction problem compatible with both contexts.

To provide a flexible multi-solver transformation, variability models are encoded into a Generic Constraint Representation (GCR). This generic representation is a solver-readable notation capable of representing constraint satisfaction problems with different constraint systems and compatible with different solvers. Thus, there is no need to encode the constraint representations into particular solver's input. In other words, GCR resembles what DIMACS representation is for SAT solvers but for different types of solvers.

Semantic domain

The semantic domain is a representation of the meaning given to a program, an expression, or model in a particular language. As described above, the meaning provided by the language semantics agrees with the community consensus. In the exercise to formally define the language semantics it is necessary to also define the semantic domain as a mathematical representation of this meaning.

The community in the variability modeling domain agrees that the semantic of variability models is the set of configurations derived from the model. Then, the semantic domain is a representation of this configuration set. A common practice is to represent the semantic domain using the mathematical definition of sets, Cartesian products, and power sets [SHTB07, DHR10, Ber12]. For instance, Schobbens *et al.* [SHTB07] defined the semantic domain $\mathcal{S}$ of feature-oriented languages in terms of their definition of *product lines* which are also defined in terms of *products*. That is, $\mathcal{S} = product\ lines$. In their definition, a product p is a set of feature names $p \in \mathcal{P}F_{names}$, an a product line pl is a set of products, $pl \in \mathcal{P}\mathcal{P}F_{names}$.

In concordance with the semantics of variability languages, Definitions 4.1, 4.2, and 4.3 present the semantic domain of HLVL's in terms of *valuations* and *configurations* as follows.

Definition 4.1 Valuation. *A valuation is a pair $\langle e_k, v\rangle$ where e_k is an element in the model, $e_k \in \mathcal{E}$, v is a variant in the set of variants associated to e_k, $v \in V_k$, $\vartheta : e_k \to V_k$. Valuations represent the match up between an element and one of its variant.*

Definition 4.2 Configuration. *A configuration C is a set of valuations holding the constraints in the model. $C = \{c \mid c \text{ is a valuation}\}$ A configuration C is full iff there is a pair $\langle e_k, v\rangle$ for each element e_k in the model, otherwise C is a partial configuration.*

Definition 4.3 HLVL's Semantic domain $\mathcal{S}$. *The semantic domain of HLVL is the set of all valid configurations (full) represented by a model. $\mathcal{S} = \mathcal{P}(C)$.*

To produce the set of all valid configurations represented by a model HLVL models are transformed into Constraint Satisfaction Problems (CSP). A CSP is defined by a set of problem variables (i.e., the unknowns) where each variable is associated with a domain of values and a set of constraints. A constraint is a logic relation between several variables restricting the values that these variables can simultaneously take. Solving a CSP consists in finding a set of *valuations* that satisfy all the constraints. Formally, a CSP is defined as follows.

Definition 4.4 Constraint Satisfaction Problem (CSP), taken from [RvBW06]. *A CSP is represented by a tuple (V, D, C) where $V = \{V_1, V_2, \ldots, V_n\}$ is a set of variables, $D = \{D_1, D_2, \ldots, D_n\}$ is a set of domains for*

the variables, and $C = \{C_1, C_2, \ldots, C_m\}$ *is a set of constraints.*

The variables in a constraint problem may range over *Booleans* when the set of values is $\{True, False\}$ or equivalently, in the integer domain $\{0..1\}$. Thus, when variables range over Booleans, the constraints are expressed using Boolean expressions. From this stand-point, solving the question whether a boolean expression is satisfiable, is equivalent to answer the question whether the corresponding Boolean CSP is consistent [Apt03]. Then, from now on SAT problems are considered CSP over Boolean constraint systems.

Semantic Mapping

The semantic mapping associates expressions in the language, or the language's abstract syntax to their meaning in the semantic domain $\mathcal{M} : \mathcal{L} \rightarrow \mathcal{S}$. Often, the semantic function inductively defines the semantic map for complex expressions in terms of the simpler ones. For instance, in their proposal, Schobbens et al. [SHTB07] start with the definition of the set of feature names P. Then, they map products to sets of feature names $c \in \mathcal{P}P$. and finally, they map product lines pl as the power set of products $pl \in \mathcal{P}\mathcal{P}P$

The semantic mapping in this thesis maps variability models in HLVL to constraint satisfaction problems. As explained above, instead of choosing a single logic representation, a single set of transformation rules, and a single type of solver, this thesis proposes using the logic representation and transformation rules that better fit a particular model.

Then, the semantic mapping uses *inference rules* [Plo81] to decide the suitable set of transformation rules. Inference rules are relationships stating that if, a set of conditions are obtained, then a set of conclusions can be deducted. This concept comes from the natural deduction in logic and proof theory. In general, inference rules are presented in the form:

$$\frac{conditions}{conclusions}$$

The following definitions are relevant to specify the conditions and conclusions in the inference rules for HLVL's operational semantics. They formally define the context of a model, the support level of a model and the context of a tool. To illustrate these definitions, consider the HLVL model of the extract of the parking assistant system (PAS) presented in Figure 4.22 and a context of implementation supporting a SAT solver and a CLP solver (*e.g.,* SAT4J and GNUProlog).

Definition 4.5 *The* **context of a model** Q_M *is the set of* HLVL(x) *sublanguages required to support the constructs in* M*.*

In the example, the context of the model is the set $Q_M = \{HLVL(bool), HLVL(att)\}$ because all the constructs in the model are contained in those sublanguages.

Definition 4.6 *The* **support level of a model** S_M *is the highest support level of the sublanguages in the context* Q_M *of the model.*

The support level of the model in the example is $S_M = Level2$ because is the highest support level in the context of the model.

```
model Parking_Assistant_System
options:
  choice memory, processor, sensors, speedSensor, positionSensor, feedback, audio,
       vibration, visual
  att integer cores domain: 1..7
  att integer size domain: [2, 8, 16, 32]
relations:
  r1: common(sensors)
  r2: group(feedback, [visual, audio, vibration], [1,2])
  r3: group(sensors, [speedSensor, positionSensor], [1,*])
  r4: decomposition(memory, [size], [1,1])
  r5: decomposition(processor, [cores], [1,1])
  r6: implies(memory, procesor)
  r7: implies(procesor, memory)
```

Figure 4.22: Example, extract of the parking assistant system (PAS) in HLVL

Definition 4.7 *The* **context of a tool** Q_T *is the set of support levels provided by the solvers available in the tool.*

The context of the tool in the example is $Q_T = \{Level1, Level2\}$ because the solvers in the tool support constraint satisfaction problems over with constraint systems over Booleans and Integers.

General Inference Rule

The following general rule examines the context of a model M, the minimum support level of the model, and the context of the tool to produce a transformation function $T(M)$, formally:

Definition 4.8 General Inference Rule. *Given a model M and its context Q_M, the set of transformation rules is decided according to the rule:*

$$\frac{S_M, Q_T}{T(M)}$$

Where

- S_M *is the support level required for* Q_M.
- Q_T *is the context of the tool.*
- $T(M)$ *is the resulting transformation function.*

The resulting transformation function $T(M)$ represents a set of transformation rules that applied to non-orthogonal constructs in HLVL produces a constraint satisfaction problem compatible with (1) the support level required by the sentences in the model; and (2) the support level provided by the tool.

Inference Rules for Non-Orthogonal Constructs

The following definitions present the six inference rules considering the minimum support required by the context of an HLVL(x) sublanguage and the context provided by a tool. The inference rules for non-orthogonal

constructs produce $T(M)$ represented as a set of R_i transformation rules. These R_i reference the collection of transformation rules characterized in the systematic mapping study presented in Chapter 2. To further information about the definition and characterization of each R_i, the reader may review Table 2.5 in Chapter 2.

Definition 4.9 Rule for Boolean models and Level 1 support. *Given a model M with $Q_M = \{HLVL(bool)\}$, $S_M = Level1$ and a tool with $Q_T = \{Level1\}$. M is a model with boolean items, FODA-related variability relations. Tools supporting Level 1 usually provide SAT and BDD solvers. The CSP encoding M is obtained by applying any of the rules in the set $T(M)$ obtained by the following rule:*

$$\frac{\{Level1\}, \{Level1\}}{T(M) \in \{R_1, R_2, \ldots, R_7, R_{24}\}}$$

Definition 4.10 Rule for Boolean models and Level 2 support. *Given a model M with $Q_M = \{HLVL(bool)\}$, $S_M = Level1$ and a tool with $Q_T = \{Level2\}$. M is a model with boolean items and FODA-related variability relations. Tools supporting Level 2 provide SMT, CLP, and CCP solvers. The CSP encoding M is obtained by applying any of the rules in the set $T(M)$ obtained by the following rule:*

$$\frac{\{Level1\}, \{Level2\}}{T(M) \in \{R_{18}, R_{22}\}}$$

Definition 4.11 Rule for Non-Boolean models and Level 1 support. *Given a model M with $Q_M = \{HLVL(bool), HLVL(att), HLVL(num), HLVL(mult),\}$, $S_M = Level2$ and a tool with $Q_T = \{Level1\}$. M is a model with boolean and non-boolean items, FODA-related variability relations, attributes, arithmetic expressions, multiplicities, and the tool supports SAT and BDD solvers. The CSP encoding M is obtained by applying any of the rules in the set $T(M)$ as follows:*

$$\frac{\{Level2\}, \{Level1\}}{T(M) \in \{R_{10,R_28}\}}$$

Definition 4.12 Rule for Non-Boolean models and Level 2 support. *Given a model M with $Q_M = \{HLVL(bool), HLVL(att), HLVL(num), HLVL(bool), HLVL(mult),\}$, $S_M = Level2$ and a tool with $Q_T = \{Level2\}$. M is a model with boolean and non-boolean items, FODA-related variability relations, attributes, arithmetic expressions, multiplicities. Tools supporting Level 2 can work with SMT, CLP, and CCP solvers. The CSP encoding M is obtained by applying any of the rules in the set $T(M)$ obtained by the following rule:*

$$\frac{\{Level2\}, \{Level2\}}{T(M) \in \{R_8, R_9, R_{11}, R_{12}, R_{16}, R_{17}, R_{18}, R_{25}, R_{26}, R_{27}\}}$$

Definition 4.13 Rule for Non-Boolean real models and Level 3 support. *Given a model M with $Q_M = \{HLVL(bool), HLVL(att), HLVL(num), HLVL(bool), HLVL(mult), HLVL(num{+}real), HLVL(att{+}real)\}$, $S_M = Level3$ and a tool with $Q_T = \{Level3\}$. M is a model with boolean and non-boolean items integer or reals, FODA-related variability relations, attributes, arithmetic expressions, multiplicities. Tools supporting Level 3 provide solvers for linear programming, mixed linear programming, and CCP supporting constraint systems over real numbers. The CSP encoding M is obtained by applying any of the rules in the set $T(M)$ obtained by the*

following rule:

$$\frac{\{Level2\}, \{Level2\}}{T(M) \in \{R_{22}, R_{23}, R_{16}\}}$$

Definition 4.14 Rule for Non-Boolean integer models and Level 4 support. *Given a model M with $Q_M = \{HLVL(bool), HLVL(att), HLVL(num), HLVL(bool), HLVL(mult), HLVL(set)\}$, $S_M = Level2$ and a tool with $Q_T = \{Level4\}$. M is a model with boolean and integer items, FODA-related variability relations, attributes, arithmetic expressions, multiplicities. Tools supporting Level 4 provide solvers for specialized constraint systems and extensions of the CCP paradigm. The CSP encoding M is obtained by applying any of the rules in the set $T(M)$ obtained by the following rule:*

$$\frac{\{Level2\}, \{Level2\}}{T(M) \in \{R_{15}, R_{20}\}}$$

To summarize, Table 4.2 maps the support levels and the tool's contexts. Each cell in the table represents one of the inference rules for non-orthogonal constructs. To illustrate how the inference rules and Table 4.2 work, recall the example presented in Figure 4.22 and its hypothetical context, *e.g.,* a tool supporting SAT4J and GNUProlog. The application of the inference rules results in $T(M) \in \{R_1, R_2, R_3\}$ since the context of the model requires a support level $S_M = Level2$ and the compatible solver is GNUProlog a Level 2 solver for constraint logic programming. Now, any framework implementation can use any of the R_i rules for the transformation. The following chapter, Chapter 5, presents the architecture and prototype of the transformation framework using different transformation rules to support analysis tasks.

Table 4.2: Mapping between support levels and tool's contexts.

Tool's Context	Support level and solvers		Level 1- Boolean	Level 2 - Integer	Level 3 - Real	Level 4 - Complex CSP
	Level 1	SAT, BDD	$T(M) \in \{R_1, R_2, \ldots, R_7, R_{24}\}$	$T(M) \in \{R_{10}, R_{28}\}$		
	Level 2	SMT, CLP, CCP-integer	$T(M) \in \{R_{22}, R_{18}\}$	$T(M) \in \{R_8, R_9, R_{11}, R_{12}, R_{16}, R_{17}, R_{18}, R_{25}, R_{26}, R_{27}\}$		
	Level 3	CCP-real, LP			$T(M) \in \{R_{22}, R_{23}, R16\}$	
	Level 4	Complex CCP		$T(M) \in \{R_{15}, R_{20}\}$		

Inference Rules for Orthogonal Constructs

HLVL has three orthogonal constructs: visibility, conditional mutex and conditional implies. The inclusion of any of those constructs in the model do not have an effect on the expressiveness of the sublanguages and may be combined with any of them. Whenever one of these constructs appears in a model, the general inference rule is applied and the resulting constraint satisfaction problem is extended with the constraints produced with respect to individual inference rules.

The inference rules for orthogonal constructs use the following notation for constraint satisfaction problems:

- Let P be the constrain satisfaction problem produced by the application of the general inference rule to a model M, that is, $T(M) = P$. P is triple $P = (V, D, C)$ where $V = \{V_1, V_2, \ldots, V_n\}$ is the set of variables, $D = \{D_1, D_2, \ldots, D_n\}$ is the set of domains associated to those variables, and $C = \{C_1, C_2, \ldots, C_m\}$ is the set of constraints.
- $P.V$, $P.D$, $P.C$ are references to the set of variables, domains, and constraints in P, respectively.
- B is a Boolean variable with domain in $\{true, false\}$
- The constraint $v =\sim$ represents that the variable v is undetermined.

Conditional Implication The conditional implication contains a constraint expression conditioning the inclusion of a set of choices. This variability relation is encoded using Boolean variables and reified constraints. Those variables and constraints are included in the CSP as shown in the following rule.

Definition 4.15 Conditional Implication Rule. *Given a model M containing the conditional implication with a constraint expression CE and a set of k choices $C_1, C_2, \ldots, C_k$, the CSP P is extended as follows:*

$$\frac{\langle \texttt{implies}(CE \ , \ [C_1, C_2, \ldots, C_k])\rangle}{P.V \cup B, \quad P.D \cup \{true, false\}, \quad P.C \wedge B \Leftrightarrow CE \quad \wedge B \Leftrightarrow C_i, \quad 1 \leq i \leq k}$$

Conditional Mutex Similarly to the previous rule, the conditional mutex also contains a constraint expression, but it conditions the exclusion of a set of choices. This variability relation is also encoded using reified constraints according to the following rule.

Definition 4.16 Conditional Mutex Rule. *Given a model M containing the conditional mutex with a constraint expression CE and a set of k choices $C_1, C_2, \ldots, C_k$, the CSP P is extended as follows:*

$$\frac{\langle \texttt{mutex}(CE \ , \ [C_1, C_2, \ldots, C_k])\rangle}{P.V \cup B, \quad P.D \cup \{true, false\}, \quad P.C \wedge B \Leftrightarrow CE \quad \wedge \neg(B \wedge C_i), \quad 1 \leq i \leq k}$$

Visibility Visibility relations hide or enable a group of options and their relations considering the entailment of a constraint expression. To encode this variability relation some Boolean variables and reified constraints are added to the CSP. Also, the following rule includes the symbol $\sim$ in the domain of each option to represent that an option is undefined.

Definition 4.17 Visibility Rule. *Given a model M containing the visibility relation with a constraint expression CE and a set of k options $O_1, O_2, \ldots, O_k$, the CSP P is extended according to the rule.*

$$\frac{\langle \texttt{visibility}(CE \ , \ [O_1, O_2, \ldots, O_k])\rangle}{P.V \cup B, \quad P.D \cup \{true, false\}, \quad P.C \quad \wedge B \Leftrightarrow CE \quad \wedge B \Leftrightarrow O_i \neq\sim \quad \wedge \quad \forall \quad c \in P.C | O_i \text{ is a variable in } c, B \Leftrightarrow c}$$

The constraint $O_i \neq\sim$ defines that if the constraint expression $CE \Leftrightarrow true$ is entailed, the option O_i must not be undefined.

4.5 Summary

This chapter unveiled the constituent elements of Coffee, the framework proposed in this thesis to ease the interoperability between tools and the coupling inside tool's implementations. The following subsections summarizes the characteristics of the contribution and address most questions formulated in Section 4.1. The remaining questions will be answer in the evaluation chapter (*cf.* Chapter 5).

4.5.1 The High-Level Variability Language

The High-Level Variability Language (HLVL) is an expressive and extensible textual language that can be used as **a modeling** and **an intermediate** language for variability. HLVL was designed following an ontological approach, *i.e.,* by defining their elements considering the meaning of the concepts existing on different variability languages. This proposal not only provides a unified language based on a comprehensive analysis of the existing ones, but also sets foundations to build tools that support different notations and their combination in a concept-driven approach.

HLVL's expressiveness

As recurrently mentioned in this dissertation, the language's expressiveness is the main concern from the variability modeling point of view. Then, the definition of the set of variability concepts supported by the language considering the trade-off between expressiveness and analysis capabilities was one of the challenges tackled in this chapter.

Two sources were considered to determine the expressiveness level of HLVL. On the one hand, there is the glossary of variability modeling concepts surveyed in the literature review and reported in Chapter 2. On the other hand, are the results and lessons learned form the ontological evaluation presented in Chapter 3. More particularly, the structural elements and variability patterns in Asadi *et al.*'s ontology [AGWH12] and the completeness and clarity metrics proposed bye Recker *et al.* [RRIG09].

Table 4.3 presents the variability concepts in the glossary as columns and HLVL(x) sublanguages in the rows. The intersections in the table map each sublanguage with the concepts using the symbols • to denote total support, and ◐ for partial support. The final row in the table summarizes all the concepts supported in HLVL and includes the support for the orthogonal variability relationships. Also, the last row contains the symbol ⍟ to highlight that the expressiveness in the conditions in guarded implication/mutex is limited to the supported expressions language. The column in grey depicts that the expressions language in HLVL does not support first-order logic, an item to be considered in future extensions.

The evaluation and discussion about HLVL's ontological expressiveness is presented in Chapter 5.

Table 4.3: Expressiveness Levels in the HLVL(x) sublanguages.
In the table, • represents total support, ◐ partial support, and ⋆ conditional support.

Language	Variability Units								Variability Relations													
	Type		Valu		Multi		Other		Common relations						Expressions			Special relations				
	Boolean	Non-Boolean	Single Valuation	Set of Valuations	Single Instance	Multiple Instances	Attributes	Attached Info	Commonality	Implication	Mutex	Hierarchy one-to-one	Hierarchy one-to-many	Propositional	First-Order	Relational	Arithmetic	Quantified implication	Hierarchy visibility	Conditioned mutex	Conditioned implication
HLVL(*bool*)	•		•		•			•	•	•	•	◐	•	•							
HLVL(*att*)							◐									◐	◐				
HLVL(*mult*)						•										◐	◐	•			
HLVL(*num*)		◐										•				◐	◐				
HLVL(*att* + *real*)							•									•	•				
HLVL(*num* + *real*)		•														•	•				
HLVL(*set*)				•																	
HLVL	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	•	•	•	⊛	⊛

4.5.2 Logical representation for variability models in HLVL

Two factors drove the design of the Coffee's analysis support. First, the decision to use constraint satisfaction problems as logic representations of variability models. Second, the hypothesis to encode variability models using an intermediate representation to enable the multi-solver support.

Although variability models in HLVL are encoded into constraint satisfaction problems, this thesis does not choose a single logic paradigm and transformation rules because the transformation considers different constraint systems. A constraint system specifies the types of variables and kinds of constraints a solver can handle in terms of sets, functions, and predicates [SR90]. Thus, constraint satisfaction problems over different constraint systems provide a flexible transformation framework. From this standpoint, a constraint satisfaction problem is a generic form for other satisfiability problems, such as SAT, SMT, and CLP, among others.

The flexible transformation framework using multiple logic representations and transformation rules was achieved by designing and implementing of the language's operational semantics following a concept-driven approach. This approach included the definition of (1) six HLVL(x) sublanguages each grouping a set of modeling concepts; (2) four support levels considering the analysis requirements of the sublanguages; (1) a set of orthogonal constructs; (3) a set of definitions to characterize the context of model and a tool in terms of its expressiveness and analysis needs; and (4) three inference rules that examine the context of the model and

the context of the tool. Then, the inference rules and decides which transformation rules should be applied to procure a constraint satisfaction problem compatible with both contexts. These four elements together conform the Coffee's context-aware transformation framework. This context-aware framework can encode a variability models in HLVL into a constraint satisfaction problem over the constraint system that meets the model's expressiveness needs.

The final design decision to provide a multi-solver solution was to define the intermediate representation used to encode the constraint satisfaction problems produced by the context-aware transformation. This intermediate representation is what this proposal called the Generic Constraint Representation (GCR). The objective to include GCR as an intermediate representation is to avoid any solver's concrete syntax, similarly to the HLCL framework. However, HLCL was not an option because it is an in-house implementation of an abstract syntax for Constraint Logic Programming.

Instead, the implementation of the framework uses MiniZinc, a solver-independent modeling language supporting most of the solvers used for reasoning about variability models [NSB+07]. Also, MiniZinc is compatible with Sat Solvers, SMT solvers, CSP solvers, supports real-number arithmetic, and constraint systems over sets. Moreover, constraint satisfaction problems modeled in MiniZinc are intuitive and readable since the syntax of the notation is simple, concise, and does not include tags or extra information.

Part III

Results Analysis, Discussion, and Outlook

Chapter 5

Evaluation, Discussion, and Outlook

This chapter presents the evaluation conducted to demonstrate the *expressiveness* of the High-Level Variability Language and the *flexibility* of the framework supporting automated analysis of variability models written in HLVL.

The following sections present (1) the ontological analysis of the expressiveness of HLVL; and (2) the feasibility of the proposal by implementing the framework and show its flexibility; and (3) the evaluation of Coffee considering the characterizations and scenarios proposed by experts in the community [ES15, tBSE19, BC19].

5.1 Ontological Analysis of the Expressiveness of HLVL

This section presents the ontological evaluation of HLVL. This evaluation aims to measure the ontological expressiveness of HLVL using the evaluation framework presented in Chapter 3. The evaluation framework applies the theory of ontological expressiveness to provide a theoretical analysis of variability modeling languages in order to determine its ability to represent variability in real-world domain models. This framework is grounded on the theory of ontological expressiveness [WW93], using a foundational ontology for variability languages combining the works of Asadi *et al.* [AGWH12] and Reinhartz-Berger *et al.* [RBSW11].

The framework in Chapter 3 is a Goal-Question-Metric (GQM) approach [BCR94]. The framework provides both, the elements to design the evaluation, the foundational ontology, and a conduction process. The following subsections present the design of the evaluation adjusted to the HLVL, the conduction of the ontological analysis, and the obtained results.

5.1.1 Design of the Evaluation

The design of the theoretical framework provides the goals, metrics, research questions, and hypothesis to evaluate the ontological expressiveness of variability modeling languages. The following are the elements of the evaluation.

Goal. The goal of the evaluation is *to evaluate HLVL with respect to its completeness and clarity from the point of view of the expressiveness in the context of an ontological analysis.*

Metrics. The metrics in this evaluation are the four measures of potential ontological deficiencies proposed by Recker *et al.* [RRIG09]. Table 3.3 presents how to calculate each measure.

M1 Degree of Deficit (DoD), calculated as the ratio ontological constructs that cannot map to any language construct.

M2 Degree of Excess (DoE), calculated as the ratio language constructs that cannot map any ontological construct.

M3 Degree of Redundancy (DoR), as the ratio of constructs in the modeling language mapping the same ontological constructs.

M4 Degree of Overlap (DoO), calculated as the ratio of language constructs mapping more than one ontological construct.

Questions. Table 5.1 presents the research questions[1], each paired with their rationale and correspondent metrics.

Table 5.1: Research questions for the ontological analysis of the HLVL.

Question	Rationale	Metric
EQ1. Does HLVL map all the constructs in the ontological model?	This question serves to determine the completeness or incompleteness of the HLVL.	DoD
EQ2. Are there any HLVL constructs that cannot be mapped into ontological constructs?	This question is related to determine if the HLVL has construct excess. Also,it contributes to elaborate an explanation about the clarity of the language.	DoE
EQ3. Is the mapping a one-to-one relation?	This question serves to determine if the HLVL has construct redundancy and construct overload. EQ3 contributes with EQ2 to conclude about the clarity of the language.	DoR, DoO

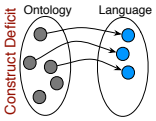
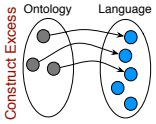
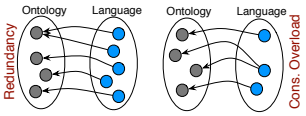
Hypothesis. The refinement of the stated questions relies on the analysis of three hypotheses, each one with null and alternative forms, related to Recker's *et al.* metrics, as synthesized in Table 5.2.

Threats to validity in the evaluation

Chapter 3 previously discusses that the validity of the results may be affected by the selection of the ontology and the mapping between the language and the ontology. As this evaluation applies the framework presented in Chapter 3 and uses the same foundational ontology, the threats regarding the selection of the ontology were addressed already. Now, to address the threats regarding the mapping between the language and the ontology this analysis was also conducted in iterative steps. It is worth noting that the design of the language was impacted by the results in Chapter 3, then the language constructs were designed to avoid the defects reported in the previous evaluation.

[1]These questions are coded with the prefix E to indicate that they are in addition to the main research questions and are addressed only in this chapter through the evaluation

Table 5.2: Hypotheses

Question	Null hypothesis	Alternative hypothesis	Defect
EQ1	$H1_0$: All ontological constructs were mapped to HLVL constructs. $H1_0 : DoD = 0\%$	$H1_1$: One or more ontological construct cannot be mapped to any HLVL construct. $H1_1 : DoD > 0\%$	
EQ2	$H2_0$: All the HLVL constructs were mapped. $H2_0 : DoE = 0\%$	$H2_1$: One or more HLVL construct cannot be mapped to any ontological constructs. $H2_1 : DoE > 0\%$	
EQ3	$H3_0$: The map is one-to-one. $H3_0 : DoR = 0\% \wedge DoO = 0\%$	$H3_1$: The map is NOT one-to-one. $H3_1 : DoR > 0\% \vee DoO > 0\%$	

5.1.2 Conduction

The mapping of the HLVL against the foundational ontology requires two mappings: the representation mapping and the interpretation mapping. The representation mapping determines whether and how ontological constructs are represented using the language constructs. The interpretation mapping determines whether and how a grammatical construct stands for a real-world construct and answer the research questions. The following subsection presents the representation mapping and interpretation mapping.

Representation mapping

The representation mapping describes first the mapping of the sources of variability followed by the mapping of variability patterns. Table 5.3 presents a summary of the representation mapping of the sources of variability following the format of Asadi's *et al.* ontology.

Mapping the Sources of Variability. As shown in Table 5.3, the sources of variability are divided in two groups: the structure elements and the process elements. The mapping of the sources in variability in the ontology is the following:

Things. The ontological model defines things as elementary units that have properties. In HLVL, variable items in a model are represented by options. Then, things are mapped to options. Options offer much possibilities to model elements in a system than the boolean variables in the mapping presented in Chapter 3.

Properties. Accordingly to the ontology, properties are linked to things, as they represent a particular characteristic of a thing. This definition is consistent to *attributes* that represent particular characteristics of variable items in a system.

Table 5.3: Representation mapping between the structural elements in the foundational ontology and HLVL constructs.

Concepts			Mapping - Rationale
Variability source	Structure	Things	**Options** - the variability units in HLVL are called *options*, they represent the variable items in a system that must be chosen or defined in a configuration process.
		Properties	**Attributes** - HLVL attributes represent properties or particular characteristics linked to options.
		Lawful state space	**Variants** in the domain of the options and attributes that comply with the variability relations in the model.
		State	**Domains** of the options and attributes. This mapping considers that the state is the set of variants for options and attributes.
		State law	**Constraint expressions** - the constraint expressin in HLVL are the rules restricting the variants in the domains of the attributes in the model.
	Process	Lawful event space	**Variability relations** - variability relations in the HLVL model causing the changes in the options' state.
		History	**Visibility** (partially) - the variability relationship provides three states: (1) deactivated, (2) activated-undefined, and (3) activated-defined.

lawful state space. The ontological model defines the lawful state space as an ontological construct defining the set of states of a thing complying with the state laws of the thing. To explain this mapping considering that things were mapped to options and properties to attributes, the mapping reviews the definitions of *state* and *state law*.

State. The state of a thing represents the possible values of a thing's attributes. In HLVL, the **domain** of an option or attribute is a set of possible values that can be associated with the option or attribute . Thus, the state of a thing maps to the set of variants in the domains of the option/attribute used for representing a thing and its attributes.

State law. The ontology defines the state law as a rule that restricts the values of the attributes of a thing. In HLVL, once defined and linked to an option, attributes can appear as operands in constraint expressions. Then, the constraint expressions are the constructs that represent the rules restricting the domains of attributes. Hence, the state laws are mapped to the constraint expressions over the attributes representing the state of a particular thing.

The conclusion is that the **lawful state space** can be mapped to the set of variants in the domain of the attributes agreeing with the constraint expressions in the model.

Lawful event space. The ontology defines this construct as the set of all events in a thing that are lawful (with respect to the state laws). The following mapping considers that an **event** is defined as a change in the state of a thing that can be internal or external. The HLVL's semantics consider two changes in the state of the options. First, the state of an option is *undefined* before the configuration. Second, the state of an option options is *defined* when the configuration determines the pairs $(option, variant)$, such pairs satisfy all the constraints in the model. Now, considering that the defined options and the pairs $(option, variant)$

meet all the variability relations in the model, the **lawful event space** is mapped to the set of variability relations in the HLVL model.

History The ontology defines history as the chronologically-ordered states that a thing traverses in time. HLVL partially maps this ontological construct using the visibility operator. The inclusion of the visibility operator introduces a new change in the state of an option.The visibility operator hides one or more options such that the options are excluded from the configuration process until the visibility condition meets. Consequently, the inclusion of the visibility operator causes the following sequence of states: (1) deactivated, (2) activated-undefined, and (3) activated-defined. The mapping is partial because the semantics of the language do not consider more states or changes in the pairs $(option, variant)$ during the configuration.

Mapping the Variability Patterns. The variability patterns are observable characteristics of the products in a variability-intensive system. The variability patterns defined in Chapter 3 were defined for Boolean variability units and the configurational semantics where a product is a set of selected variable items represented by their names. To be consistent to the configurational semantics of HLVL, the following mapping requires adjustments in the definition of *equivalence* as follows.

Definition 5.1 Equivalence. *Let S and T be two products. S is equivalent to T ($S \equiv T$) iff there is a mapping between S and T where $\forall$ pair $(option, variant) \in S$ there is one, and only one, equivalent pair $(option, variant) \in T$*

Also, to provide a better readability of the following mapping, recall the definition of similarity from Chapter 3.

Definition 3.2 Similarity. S is similar to T with respect to an equivalence subset p, denoted as $S \cong_p T$, iff there exists S', T' such as $S' \subset S$ and $T' \subset T$, then $S' \equiv T'$. In other words, the concept of similarity refers to elements that are common to products in a product line.

Table 5.4 presents a summary of the representation mapping of the variability patterns in Asadi's *et al.* ontology. The mapping of the four variability patterns considers that these patterns should be observable excluding the set of common or core variable items in a variability model. Then. the mapping excludes the constructs `common` and `decomposition` with cardinality `[1,1]`, as they can be used to describe commonality.

Full Similarity One-Side. Two products S, T are full similarly one-side when they satisfy the similarity relation, and an equivalence relation can be established, in such a way that a subset of S is equivalent to T or *vice versa*. This pattern can be mapped to the HLVL constructs that associate two or more options where the association of one option to a particular variant, defines the valuations in the other options. Then, the variability relations `decomposition` and `implies` map the pattern.

Partial Similarity. Two products are partially similar when they have equivalent subsets of pairs $(option, variant)$. This pattern can be mapped to constructs in HLVL that produce products sharing subsets of equivalent pairs $(option, variant)$, different from the core options. The construct `group` representing hierarchical relations between one parent and many children with cardinality different than $[1, 1]$ map to this pattern. Partial similarity can also be observed inside the options defined using the construct `set`, as they can be

Table 5.4: Representation mapping between the variability patterns in the foundational ontology and HLVL constructs.

Pattern	Mapping - Rationale
Full Similarity One-Side	`decomposition`, `implies` - The inclusion of these constructs in a model makes it possible to find products with the full similarity one-side property in the set of solutions.
Partial Similarity	`group` with cardinality `[n,m]` and options declares with the keyword `set` - These constructs produce products sharing subsets of equivalent pairs $(option, variant)$, different from the core options. Also, the pattern may be found among the sets of variants linked to the options defined using the construct `set`.
Dissimilarity	`group` with cardinality `[1,1]`, `mutex` , and options declared as `enum` - These constructs produce products that do not share options or with options paired to different variants in the enumeration's domain.
Ordering	No construct in HLVL can map this ontological construct.

bounded to one or more variants in the domain during the configuration process.

Dissimilarity. Two products are completely dissimilar if no similarity relation can be established. The reader may consider that, there is no two completely dissimilar products because variability models contain a set of common or core variable items. This pattern can be mapped to the `group` with cardinality `[1,1]`, `mutex` and `enum` constructs that represent hierarchical one-to-many relations, mutual exclusion, and options declared as enumerations, respectively. These constructs produce products that do not share options or with options paired to different variants in the enumeration's domain.

Ordering The ordering pattern occurs when two products have a similarity relation but they differ by an order relation. This pattern cannot be mapped to any variability relation in HLVL since none of the constructs cause an order relation in the sets of pairs $(option, variant)$ representing the products configured from a variability model in HLVL. Moreover, under the concurrent constraint programming model, it is not possible to establish an order relation over the set of valuations solving a CCP [SR90].

Interpretation mapping

The interpretation mapping determines whether and how a grammatical construct stands for a real-world construct. Table 5.5 presents the interpretation mapping. The rows in the table represent the constructs in HLVL excluding the constructs used to describe commonality. The columns in the table represent the ontological constructs. The light gray cells in Table 5.5 represents a mapping. The cells may contain a bullet to denote full support of an ontological construct and a half-empty bullet for partially support an ontological construct. The last column in in Table 5.5 in dark gray highlights that there is no language support for the the ordering pattern.

Table 5.5: Interpretation mapping between ontological constructs and HLVL constructs.
In the table, • represents a mapping, ◐ partial mapping.

Constructs	Variability Sources					Variability Patterns			
	Things	Properties	Lawful state space	Lawful event space	History	Full similarity one-side	Partial similarity	Dissimilarity	Ordering
Options	•								
Attributes - `att`		•							
Domains			•						
Variants			•						
Constraint expressions `expression`				•					
Visibility `visibility`					◐				
Hierarchy one-to-one `decomposition`						•			
Implication `implies`						•			
Hierarchy one-to-many `group`							◐	◐	
Options- `set`							•		
Mutex `mutex`								•	
Options- `enum`								•	

Measuring the potential ontological deficiencies

The potential ontological deficiencies of HLVL are measured using the interpretation mapping depicted in Table 5.5 and Recker *et al.*'s metrics [RRIG09]. The following calculations consider that the number of ontological constructs is nine and the number of language constructs is twelve.

M1: Degree of Deficit (DoD). M1 is calculated by dividing the number of not mapped ontological constructs over the total number of ontological constructs as follows:

$$DoD = \frac{\#not\ mapped\ ontological\ constructs}{\#ontological\ constructs} = 0.11$$

The number of not mapped ontological constructs is one because HLVL does not have constructs mapping the *variability pattern for ordering*. Consequently, HLVL exhibits 11% of the degree of deficit ($1/9 \times 100$). Following Recker *et al.*'s proposal, the complement of the DoD represents the level of ontological completeness. Then, HLVL's completeness level is 89%.

M2: Degree of Excess (DoE). M2 is calculated by dividing the number of not mapped language constructs over the total number of language constructs as follows:

$$DoE = \frac{\#not\ mapped\ language\ constructs}{\#language\ constructs} = 0$$

All the language constructs were mapped. Then, HLVL has zero degree of excess (DoE 0%).

M3: Degree of Redundancy (DoR). To calculate M3 we divide the number of language constructs mapping the same ontological construct over the total number of language constructs as follows:

$$DoR = \frac{\#lang.const.mapping\ the\ same\ ont.const.}{\#language\ constructs} = 0.66$$

To obtain the number of language constructs mapping the same ontological construct, we count the number of columns in Table5.5 containing more than one mapping (or partial mapping). The following is the list of redundant mappings.

1. Two language constructs map the lawful state-space: Domains and variants.
2. Two language constructs map the full-similarity one-side: **`decomposition`** and **`implies`**.
3. Two language constructs map the partial similarity: **`group`** and **`option`**.
4. Two language constructs map the dissimilarity: **`mutex`** and **`enum`**.

There are eight constructs (four pairs) mapping the same ontological construct. Then, to compute M3, we divide eight over twelve, the total number of language constructs. Consequently, HLVL's degree of redundancy is 66%.

M4: Degree of Overlap (DoO). M4 is calculated by dividing the number of language constructs mapping more than one ontological construct over the total number of language constructs as follows:

$$DoO = \frac{\#lang.const.mapping\ many\ ont.const.}{\#languageconstructs} = 0.08$$

There is only one language construct mapping more than one ontological construct. Table 5.5 shows that the **`group`** construct maps the partial similarity and the dissimilarity patterns. Then, considering the twelve language constructs, HLVL's degree of overlap is 8%.

5.1.3 Results and Answering the Evaluation Questions

The first step to answer the questions and review the hypothesis of the evaluation is to apply the defined metrics and calculate the degree of the ontological defects present in the HLVL language, Table 5.6 presents those results in the column labeled HLVL. The second step is to answer the questions in this evaluation. Each answer also includes a comparison to the results obtained in the ontological evaluation in Chapter 3 depicted in the table under the label `HLCL`.

Table 5.6: Metrics to measure the ontological defects in HLVL.

Metric	Formula	HLVL	HLCL
Degree of Deficit (DoD)	$DoD = \frac{\#not\ mapped\ ontological\ constructs}{\#ontological\ constructs}$	11%	22%
Degree of Excess (DoE)	$DoE = \frac{\#not\ mapped\ language\ constructs}{\#language\ constructs}$	0%	0%
Degree of Redundancy (DoR)	$DoR = \frac{\#lang.const.mapping\ the\ same\ ont.const.}{\#language\ constructs}$	66%	50%
Degree of Overlap (DoO)	$DoO = \frac{\#lang.const.mapping\ many\ ont.const.}{\#languageconstructs}$	8%	50%

EQ1: Does HLVL map all the constructs in the ontological model?

Three elements are relevant to answer this question: the results in the representation mapping, the results in the interpretation mapping, and the degree of deficit. The representation mapping, more particular the mapping of the variability patterns in Table 5.4 shows HLVL's construct deficit as it does not provide a construct mapping the ordering pattern. The interpretation mapping confirms the language's construct deficit as depicted in Table 5.5 where the last column does not have any mapping. Accordingly to the Recker *et al.*'s metrics depicted in Table 5.6, the construct deficit in HLVL is 11%.

Thorough the `visibility` operator, the modelers are now capable to describe relations where a condition is enforced *before* a set of options and their relations are considered for configuration. However, these constraints do not establish an order relation in the set of pairs $(option, variant)$ that represent a product. This limitation in the language is consistent to the limitations in the formalism used to provide its configurational semantics. Since, the concurrent constraint programming model does not support an order or partial order relation among the set of valuations solving a CCP [SR90].

EQ2: Are there any HLVL constructs that cannot be mapped into ontological constructs?

All the language constructs, except the ones used to define commonality, were mapped to one or more elements in the ontology proposed by Asadi *et al.* [AGWH12]. Thus, HLVL degree of excess is 0%.

EQ3: Is the mapping a one-to-one relation?

The mapping of ontological constructs to language constructs is not one-to-one for the following reasons. First, there is a difference of three constructs between the language and the ontology, as the ontology has nine constructs and the language has twelve. Then, it is not possible to have a one-to-one mapping between HLVL and the ontology, and zero degree of excess at the same time. Second, Table 5.5 shows four cases where two language constructs map a single ontological construct (*e.g.,* domains, variants → lawful state space). Thus, HLVL exhibits a 66% of degree of redundancy. Third, Table 5.5 shows that one language construct maps two ontological constructs, *i.e.,* `group` →partial similarity, dissimilarity. Then, the language has an eight percent of degree of overlap. The later results causes two ontological defects in HLVL: construct redundancy, and construct overload.

Ontological Completeness and Clarity

The completeness and the clarity of the language are measured using the potential ontological deficiencies obtained from Recker *et al.*'s metrics and calculated on Section 5.1.2. On the one hand, the Degree of Deficit (DoD) measures the level of ontological incompleteness in a conceptual modeling language. Then, the lower the DoD, the higher the level of ontological completeness. Accordingly to the results in Table 5.6, the completeness of HLVL is 89%. This result means that the language closely represents the general principles of variability under the Asadi *et al.* ontological framework.

On the other hand, the Degrees of Excess (DoE), Redundancy (DoR) and Overlap (DoO) measure the clarity of the language. The results are as follows:

1. HLVL does not have any degree of excess (DoE 0%).
2. The degrees of redundancy is 66%.
3. The degree of overlaping is 8%.

On the one hand, a low DoE is a desirable situation as it prevents user confusion due to the need to ascribe meaning to constructs that do not appear to have real-world meaning. On the other hand, the levels of redundancy and overload indicate that HLCL might be unclear and will produce potentially ambiguous representations of real-world domains.

5.2 Flexibility in the Coffee's Transformation Framework

This section presents the implementation of the three layers in Coffee and how those layers interact to produce a flexible and multi-solver transformation framework. To describe the implementation proposal, the reader may recall the conceptual model of the framework introduced in Chapter 3 and depicted again in Figure 5.1.

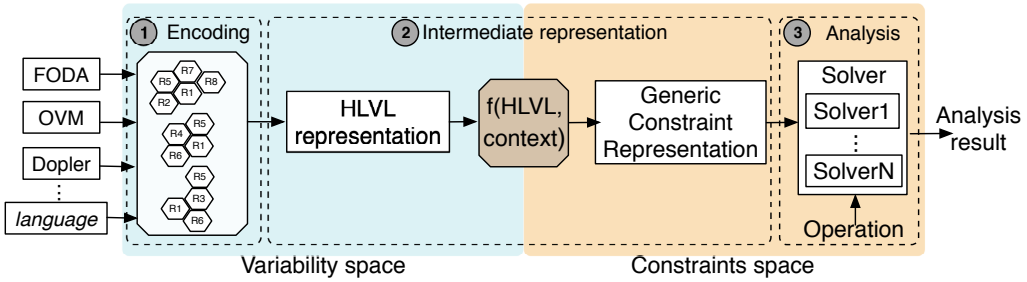


Figure 5.1: Coffee Framework - Conceptual model

Chapter 3 presented Coffee conceptually divided in two spaces, the *variability space* and the *constraints space*. These spaces contain artifacts focusing in one of the two concerns: variability modeling and variability analysis supported by constraint programming. The communication between those two spaces is called the *transition step*. This transition step represents the transformation function that produces a constraint satisfaction problem from a variability in HLVL with respect to the context.

For implementation purposes, the framework is divided in three layers, each representing one step in the conceptual model. The three layers in the implementation are: the *encoding layer*, the *intermediate layer*, and the *analysis layer*. The following subsections describe the design and implementation of each module and its contained layers. This section closes with a description of the workflow in the three layers of Coffee. The software produced in the implementation can be found in the GitHub repository: https://github.com/orgs/coffeeframework/repositories

5.2.1 The Encoding Layer

The encoding layer provides the libraries and tools to obtain an HLVL representation from variability models written in a machine-readable format. Figure 5.2 presents the structure of the implementation of the encoding layer in the module called `ModelParsers`. This module contains two sets of components divided into *abstract libraries* and *concrete parsers*.

The abstract libraries are the components implementing the transformation patterns compatible to the variability concepts in the HLVL(x) sublanguages. In this sense, the abstract libraries must be instantiated considering the expressiveness of the source language. To guide the engineering process, the components in the abstract libraries share the names with the HLVL(x) sublanguages. Then, the first step to implement a concrete parser is to identify the mapping between the source language and the corespondent HLVL(x) sublanguage. The main idea is to abstract the programmer of the equivalences between the concepts in the product line notations and the HLVL constructs. At the same time, to provide a toolkit of transformation patterns that are ready to instantiate and use in their own parser. In this sense, the programmer should be more focused on working in the processing of the variability language that they already knows, instead of a variability language that is recently learning. Then, before starting to code a parser the programmer must decide which HLVL(x) sublanguage is best suited for their needs.

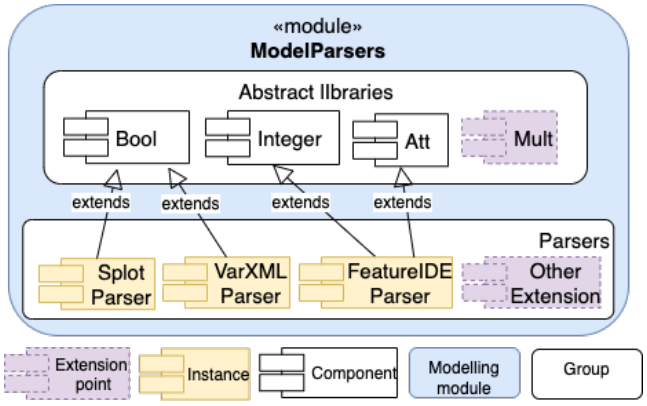


Figure 5.2: Structure of the encoding layer.

The concrete parsers in Figure 5.2 are examples of implementations available at the GitHub repository. These concrete parsers extend one or more abstract libraries. Each concrete parser has its own project in the repository with a set of unitary test wrapped in a maven project. Figure 5.2 depicts two extension points. The extension points in the structure of the `ModelParsers` module consider the changes produced in the module caused either

by the extension of HLVL or by the implementation of a new parser, *i.e.,* to support a new tool. An extension in the language implies also the creation or the modification of the abstract libraries to support the new language constructs. Likewise, the exchange of Coffee with a new modeling tool produces the creation of a new parser.

5.2.2 The Intermediate Representation Layer

In the intermediate representation layer resides the implementation of the two intermediate languages and the transition step. This layer is implemented in a module called the `HLVLImplementation`. This module contains (1) the components implementing the HLVL's infrastructure for specifying variability models,*e.g.,* syntax, editor, among others; and (2) the components implementing the context-aware transformation of HLVL models into Generic Constraint Representation (GCR).

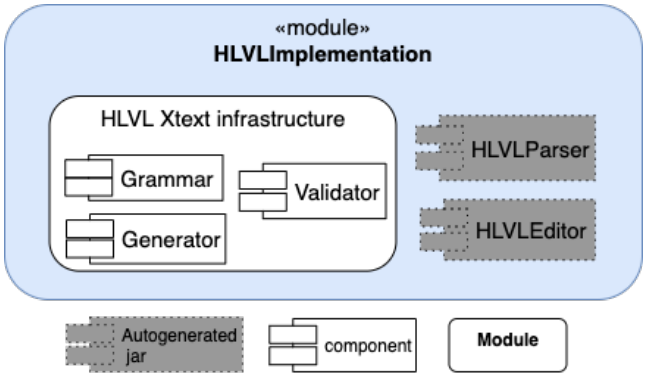


Figure 5.3: Structure of the intermediate representation layer.

The components in this layer are implemented or auto-generated using Xtext, Xtend and Java technologies. Then, the structure of this layer is consistent with the architecture of Xtext projects. The most important component in the `HLVLImplementation` is the `Generator`, since this is the component in charge of applying the context-aware transformation to produce the GCR code. The GCR representation is crucial to provide a multi-solver solution as it is allow the framework to avoid any solver's concrete syntax.

The current implementation of the framework uses MiniZinc as the implementation of the GCR. MiniZinc is a solver-independent modeling language supporting most of the solvers involved in variability management and reported in Chapter 2. Moreover, MiniZinc is compatible with Sat Solvers, SMT solvers, CSP solvers, supports real-number arithmetic, and constraint systems over sets. Also, constraint satisfaction problems modeled in MiniZinc are intuitive and readable since the syntax of the notation is simple, concise, and does not include tags or extra information.

5.2.3 The Analysis Layer

The analysis layer gathers the components and libraries to analyze a variability model described in GCR. These component is responsible for collecting the GCR implementation and the context of the available solvers in order to decide a suitable solver to perform the analysis tasks. Figure 5.4 presents the structure of the implementation of the analysis layer in the module called `Analysis`.

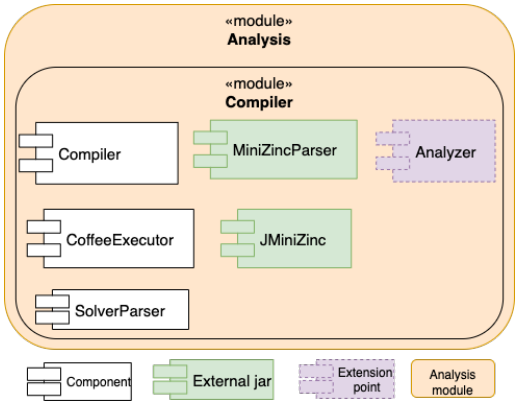


Figure 5.4: Structure of the analysis layer.

Figure 5.4 shows how the module groups the components into implemented components, external libraries and one extension point. The implemented components group contains a `Compiler`, an `Executor`, and a `SolverParser` that are in charge to decide the solver considering the context, execute the MiniZinc tools and process the output to obtain the analysis output. The external libraries in the `Analysis` module allow the execution of the MiniZinc tool-chain with the solver selected by the compiler, the GCR representation, and the configuration parameters. These libraries are available at `https://github.com/siemens/JMiniZinc`. The extension point, is designed to implement more analysis operations. Currently, the support of analysis operations in Coffee is limited to determine if a model is valid, enumerate solutions, and configure and evaluate partial configurations.

5.2.4 Workflow: from Modeling to Analysis

Figure 5.5 depicts the path of a variability model from the very beginning of the modeling process in the encoding layer to obtain an analysis output in the analysis layer.

The workflow starts by producing a variability model in HLVL, the first language in Coffee. The model can be produced by (1) encoding a model specified in an external variability language; (2) using a text editor and modeling directly in HLVL. Figure 5.5 depicts the model in HLVL as a file with `hlvl` extension. However, using the Xtext infrastructure, the model may be produced in abstract syntax in the EMF exchange format.

The second step in the workflow is to process the model using the `HLVLParser` component. This component applies the transformation rules and produces the logic representation of the variability model. This logic representation is written in the GCR, the second intermediate representation in Coffee. The GCR representation is embedded in a `json` file to be transferred to the analysis layer.

The Third step starts with the variability model in GCR, and the solving context recorded in a `json` file. The solving context is the information concerning the available solvers, *i.e.,* already installed an configured, to perform analysis operations. Those two files are the input for the `compiler` to decide which solver will be used in the next step. Then, the compiler calls the `executor` to call the external libraries and execute the MiniZinc chain, *i.e.,* FlatZinc and solver. The output is then processed by the `solverParser` component that produces

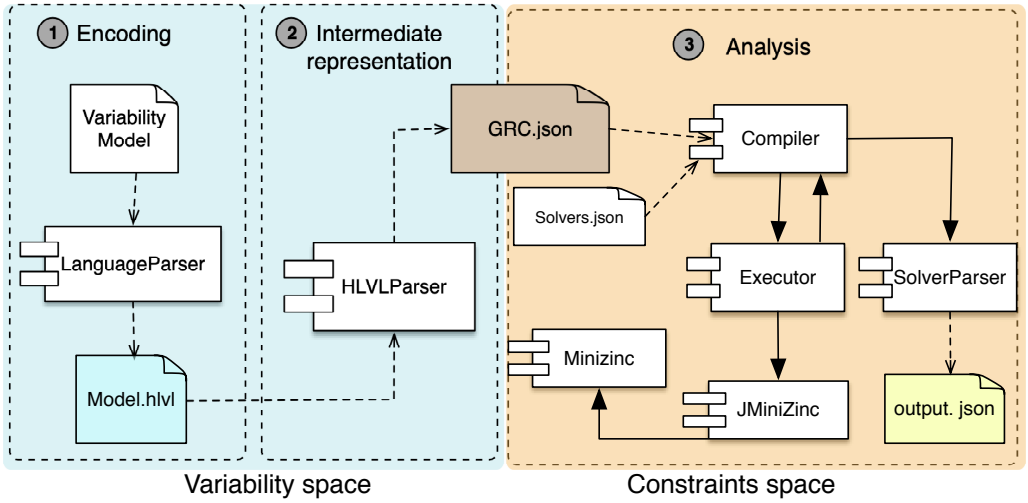


Figure 5.5: Coffee Framework - Workflow

the final analysis output, again using the `json` exchange format.

Why there is no solver depicted in the scheme of the `Analysis` layer?

Solvers are a key part in the variability analysis. However, this approach does not require one particular solver. Moreover, solvers were not included as a component in the framework as they are considered part of the context where the framework is executed. The information of the context, that is, the register of the available solvers and their characteristics is recorded using a text file in `json` format.

5.2.5 Evaluation

The evaluation of the implementation considered functional testing. Functional testing is one of the design evaluation methods proposed by Hevner *et al.*to evaluate design science artifacts [HMPR04]. The development of the software components followed the Test-Driven-Development (TDD) process. Then, each piece of software has a set of unitary tests most of them implemented using the JUnit framework. The latter because the implementation used Java and Maven technologies.

The functional tests included the implementation of transformation rules for models with context $Q_M = \{HLVL(bool), HLVL(att), HLVL(num)\}$ and a support level $Q_T = \{Level1, Level2\}$. Then, the models in the test cases for these functional tests may contain boolean and non-boolean options. Also, the test environment considered the implementation and configuration of three solvers: Gecode, Sat4J and PicatSat. Table 5.7 presents the list of repositories with the available code. Each row in the table presents the implementation layer, the name of the component, a description and the url of the GitHub repository.

Demo and test-cases

The last row in Table 5.7 presents the repository containing the test cases and documentation to execute a demo of the Coffee framework.

Table 5.7: Repositories

Layer	Component - Type	Description	Available at
Encoding layer	Bool, abstract library	Software component implementing the transformation patterns to parse into HLVL models described using tools supporting the HLVL(*bool*) sublanguage	https://github.com/coffeeframework/BasicHlvlPackage
	Integer, abstract library	Software component implementing the transformation patterns to parse into HLVL models described using tools supporting the HLVL(*bool*), HLVL(*num*), HLVL(*att*) sublanguages	https://github.com/coffeeframework/AttHlvlPackage
Intermediate layer	HLVL grammar, validator and generator	This is the software component implementing the syntax and semantics of HLVL. This component uses Xtext technologies. Therefore, it complies with the Xtext architecture for domain-specific languages. The component provides an eclipse-based editor, a stand-alone parser, and a maven package to integrate the language in other tools such as web applications.	https://github.com/coffeeframework/HLVL
Analysis layer	Reasoning, compiler, executor	This is the software component implementing the analysis layer. This component wraps the compiler, the executor, and the libraries to use the MiniZinc tool-chain to provide a generic constraint representation and multi-solver support.	https://github.com/coffeeframework/ReasoningModel
Demo	Coffee's Demonstration	This repository contains the test-cases and documentation to run a demonstration of the framework.	https://github.com/coffeeframework/DemoCoffee

5.3 Coffee Under Different Eyes

The following subsections present two comparisons of this proposal using the characterizations proposed by other authors [ES15, tBSE19, BC19].

5.3.1 Comparison of HLVL and other textual languages

The following comparison applies a classification scheme to characterize textual languages form the literature reviews proposed by Eichelberger & Schmidt's and ter Beek *et al.*. The classification scheme was first introduced by Eichelberger & Schmidt in [ES15] and later updated by ter Beek, Schmidt, and Eichelberger in [tBSE19].

Table 5.8 shows a comparison of the main capabilities of the textual variability modeling languages selected by ter Beek *et al.* in [tBSE19]. This comparison considers the level of support of the languages for the following dimensions: configurable elements, constraint support, configuration support, scalability support, and language characteristics.

Table 5.8: Comparison of the main capabilities of textual variability modeling languages

	forms of variation							data types			constraint expressions					configurations					
Language	optional	alternative	multiple	extension	attached info	cardinalities	references	predefined	derived	user-defined	simple	propositional	first-order	relational	arithmetic	default values	assign values	partial	complete	composition	formal semantics
FDL	+	+	+	−	−	∗	−	−	−	−	+	−	−	−	−	+	−	−	−	−	+
Forfamel	+	+	+	+	+	+	+	−	−	−	−	+	+	+	+	−	+	−	+	−	∗
Tree Grammars	+	+	+	−	−	+	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−
VSL	+	+	+	+	+	+	+	+	−	+	+	?	?	−	?	+	+	+	+	+	−
SXFM	+	+	+	−	−	+	−	−	−	−	−	?	−	−	−	−	−	−	−	−	−
FAMILIAR	+	+	+	−	−	−	∗	+	+	−	−	+	−	−	−	−	+	+	+	+	−
TVL	+	+	+	?	+	+	+	+	−	+	−	+	−	+	+	−	+	−	−	+	∗
CLAFER	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	−	+	+	+	+	∗
VELVET	∗	+	+	+	+	+	+	+	−	−	−	+	−	+	−	+	+	+	∗	+	−
IVML	+	+	+	+	+	∗	+	+	+	−	−	+	+	+	+	+	+	+	∗	−	−
PYFML	+	+	+	−	+	∗	−	−	−	+	+	+	−	+	+	+	+	+	−	−	−
VM	+	+	+	−	+	+	−	+	−	−	+	+	−	+	+	+	+	+	∗	+	−
HLVL	+	+	+	+	+	+	+	+	−	−	+	+	−	+	+	−	+	+	+	∗	+

+; Direct support, ∗: Indirect support, ?: Unclear support, −: No support.

Configurable elements include optional and alternative variability, multiple selection of features, feature modeling extension, attached information to the basic variability unit (usually feature attributes), feature and group cardinalities, references to other configurable elements, and data types (either predefined, derived or user-

defined). Constraint expressions support deals with the level of expressiveness, from simple dependencies, and propositional logic, to first-order logic and extensions such as relational or arithmetic expressions. Configurations has to do with both capabilities of setting values (default values and value assignment) and levels of configuration (partial and complete). Finally, scalability is provided via composition whereas the language characteristics described by formal semantics. Table 5.8 also shows that HLVL supports all of the dimensions. It only lacks support on derived and user-defined data types, first-order logic, and the definition of default values in the configuration.

Note that HLVL's scalability support via composition is highlighted in green and classified as indirect because this capability of the language is currently explored in a satellite project conducted by two students under the direction of Angela Villota. The project is discussed in the future work section in Chapter 6.

5.3.2 Applicability and Usefulness of Coffee

This section contrast Coffee against the requirements and scenarios defined by Berger & Collet [BC19].These scenarios and requirements were defined for a unified feature notation and its tool support. Yet the language proposed in this thesis is multiparadigm, this comparison serves to evaluate the applicability and usefulness of HLVL as exchange language and Coffee as framework to communicate different variability management tools.

Berger & Collet propose 14 usage scenarios, each associated to a set of requirements. Table 5.9 presents the scenarios, their requirements and a rationale of how Coffee satisfies, or not each requirement. Some of these scenarios replicate requirements, hence they were included once in the table.

Table 5.9: Fulfillment of the usage scenarios' requirements for a unified notation with HLVL.

Usage Scenario	Requirement	HLVL	Satisfied
1. Exchange	. Serializable concrete syntax.	HLVL does provide a serializable concrete syntax	●
	. Documented abstract and concreted syntax to realize importers and exporters.	The concrete syntax of HLVL has a formal definition in EBNF. Also, Coffee provides the definition and description of the HLVL(x) sublanguages and a mapping between them and the encoding libraries	●
	. Library for serialization/deserialization for tool vendors.	Coffee provides a set of abstract libraries in Java language to encode variabiity models into HLVL, but not backwards. Also, the repository offers the four concrete libraries used in the evaluation	◐
	. Extensibility of the language and information about its extensions.	The artifacts in the definition and implementation of HLVL allow its extension.	●
	. Concepts to store tool-specific data.	The language provides the `comment` to link external information to the model specification.	●
	. Independency of the tool-specific data.	The parsing of comments is already considered on the encoding library.	●
2. Storage	. Abstract syntax definition in a metamodeling notation for automated processing.	HLVL provides a formal syntax in EBNF that together with its Xtext implementation provides the EMF metamodel of the language	●
	. Concise and succinct textual syntax.	The language has programming-like syntax without the verbosity of other XML-based syntaxes	●

Table 5.9: Fulfillment of the usage scenarios' requirements for a unified notation with HLVL.

Usage Scenario	Requirement	HLVL	Satisfied
	. Textual syntax with common technology for defining concrete syntaxes (e.g. ANTLR, Xtext).	HLVL provides an Xtext implementation.	●
3. Teaching and learning	. Typical visual concrete syntax of feature models.	The language support feature modeling does not provide a visual syntax.	○
	. Provision of feature models examples (real-world and toy models).	The language definition and implementation comes with a set of toy models, but does not provide real-world examples.	◐
	. Concrete textual notation to illustrate how to scale models.	Scalability of models is a work in progress	◐
4. Writing, reading, editing	. Simple and human-readable textual concrete syntax.	The language's concrete syntax resembles the syntax of declarative programming languages where elements and relations are the most important concepts.	●
	. Language definition independent of a particular generation technology.	The Xtext implementation of HLVL allows the framework to provide: (1) textual editors with syntax highlighting and autocomplete functions (2) standard textual editors and a stand-alone parser (3) use the editor as an Eclipse or IntelliJ plugin compatibility; and (4) EMF/Ecore dependency	●
	. Use of standard text editors.		●
	. Model instances editable in standard IDEs (e.g. Eclipse, IntelliJ, IDEA, . . .).		●
	. Language's parser easy to integrate in other tool chains.	The encoding layer in Coffee provides parsers in Java	●
5. Model generation	. Translation of the complete semantics into a representation in formal language.	The framework provides a formal syntax and semantics	●
	. Generation of instances in the original language's syntax.	Coffee does not provide a tool to generate variability models for sampling or testing	○
	. Interactive instance generation, showing conflicting constraints and counter-examples.		○
6. Domain modeling	. Conventions and defaults in the language.	HLVL includes options and variability relations frequently used in variability modeling.	●
	. Textual syntax inspired by existing languages.	The syntax of the language resembles the syntax of declarative programming languages.	●
7. Configuration	. Adequate syntax for configurations by non-technical stakeholders.	The framework allows the input of list of pairs (option/attributes, variants/values) and therefore, it supports partial configuration	●
	. Configuration based on selection of features as well as value selection.		●
	. Support partial configuration.		●
	. Default configurations or exemplary configurations.	The framework does not provide default configurations	○
	. Support by inference engine requiring different types of constraints.	The language provides the visibility constraints that translated to reified constraints may drive the configuration process.	◐
8. Benchmarking	. Designed for tool support, and several implementations available.	Xtext implementation with EM metamodels with great tool support.	●

Table 5.9: Fulfillment of the usage scenarios' requirements for a unified notation with HLVL.

Usage Scenario	Requirement	HLVL	Satisfied
	. Setup to compare tool support execution times of operations in isolation.	Coffee's design contains four independent layers with separation of concerns.	●
	. Well-engineered and specified syntax and semantics of the language.	The language provides the formalization of the operational semantics and the implementation in Xtext relies on language engineering tools such ANTLR and EMF/Ecore.	●
	. Agreement on the specification of certain feature model operations.	Not supported	○
	. Availability of realistic models.	No, the framework provides a set of toy and academic examples	○
9.Testing	. Concepts to represent partial or complete configurations.	Coffee supports partial and complete configuration	●
	. Consistency checking by the language infrastructure for the configuration information.	Not supported.	○
	. Concepts capturing further testing-relevant information.	Not supported	○
10. Analyses	. Different solver strategies depending on the kinds of analyses and the language constructs.	The transformation in Coffee is compatible with different solvers and logic representations.	●
	. Language syntax and semantics, with abstractions into the logic representations for solvers.	The semantics of HLVL models are compatible to different logic representations and solvers	●
11. Mapping to implementation	. Modifier/Keyword to differentiate abstract and concrete features.	Not supported .	○
	. Well-defined mapping language.	The mapping does not consider abstract options	◐
	. Avoid common limitations (e.g. features without children and concrete, otherwise abstract).	The mapping of feature models does nos include assumptions in the structure of the model	●
12. Decomposition and composition	. Prioritized list of composition mechanisms from the literature.	In progress, developed in a project supervised by the author.	●
	. Simple mechanism easy to implement.	Prototype for merge two HLVL models.	●
	. Dedicated support for interface feature models.	Language extension to support model composition	●
13. Model weaving	. Static design-time or dynamic run-time model weaving.	The Xtext implementation of the language produce stand-alone tools to integrate HLVL to other Java applications. Also, the placing of the HLVL parser in a Docker container with an REST API allows the integration to other tools	●
	. List of variability mechanisms from the literature.	The language was designed using the glossary of variability concepts as reference	●
	. Understanding of the effort for realizing the mechanism for different types of assets.	Coffee does not support the mapping of options to implementation elements.	○

Table 5.9: Fulfillment of the usage scenarios' requirements for a unified notation with HLVL.

Usage Scenario	Requirement	HLVL	Satisfied
14. Reverse engineering	▪ Sufficient expressiveness to model real-world variability.	HLVL provides boolean and non-Boolean options to model variability in complex systems	●
	▪ Traceability and debugging information.	All elements in the language has identifiers to provide traceability after the transformation.	●

5.4 Summary

This chapter presented the evaluation of Coffee. The evaluation aimed to demonstrate the expressiveness of the variability modeling language and the flexibility of the transformation framework.

The first part of the evaluation, in Section 5.1, continues the evaluation left in Chapter 4 about the expressiveness of the language. In this chapter, the evaluation consist of applying the theoretical framework developed in Chapter 3 to conduct an ontological analysis over HLVL.

To close, Section 5.3 presents the evaluation of Coffee considering the characterizations and scenarios proposed by experts in the community [ES15, tBSE19, BC19]. First, Section 5.3.1 presents a comparison of HLVL to other textual variability languages using the characterization of ter Beek, Schmidt and Eichelberger [ES15, tBSE19]. Second, Section 5.3.2 presents the evaluation of the applicability and usefulness of the framework considering the scenarios and requirements to define an standard language presented by Berger & Collet and validated by the MODEVAR community [BC19].

Chapter 6

Concluding Remarks and future Work

"*C'est le temps que tu as perdu pour ta rose qui fait ta rose si importante.*"
Le Petit Prince, Antoine de Saint-Exupèry.

6.1 A Summary of the Dissertation

This dissertation presents the conduction of a research developed to explore the usage of intermediate languages to ease the interoperability of variability management tools. The main goal of the research was to design a constraint-based framework that supports an *expressive* variability language and a *flexible* automated analysis mechanism. The results and contributions of the research conform a framework named Coffee.

Coffee is a constraint-based framework that supports variability modeling and reasoning about variability models, two of the main tasks in the management of variability-intensive systems. This thesis addresses the problems regarding the interoperability between variability management tools, the diversity among variability modeling languages, and the strong dependencies in the automated analysis of variability models. To solve these problems using intermediate languages, this thesis presents two original contributions.

First, this thesis defines, formalizes, implements, and evaluates the High-Level Variability Language (HLVL). HLVL is an expressive textual language that unifies the modeling concepts from different modeling paradigms. Second, this research includes a proposal of a context-aware transformation framework to provide flexible, multi-language, and multi-solver support for automated analysis of variability models.

The initial steps of the research project contemplated the possibility to adapt or extend the state-of-the-art `HLCL` framework as it is the core of early implementations of the VariaMos tool-suite [MMFR+15]. More importantly, the development of the `HLCL` framework is consistent to the hypothesis of the applicability of intermediate representations developed in this thesis.

To make a decision about this adaptation or extension, an evaluation of the framework regarding its expressiveness and flexibility was in order. As reported in Chapter 3, the evaluation included the engineering of a variability management tool, and the evaluation of the HLCL's expressiveness from a theoretical point of view. The theoretical evaluation required the definition of an evaluation framework capable of answering questions, such as, *Which are the characteristics of an expressive variability modeling language?* and *How to measure the expressiveness in variability modeling languages?*. To tackle this challenge, this thesis composed an

evaluation framework grounded on the theory of ontological expressiveness [WW93], its extension to variability modeling languages [AGWH12], and the Recker *et al.*'s metrics to measure the ontological defects.

The results in the evaluation demonstrated many problematic situations with the `HLCL` framework. The ontological evaluation showed the `HLCL` weakness as variability modeling language, and the practical evaluation showed that the transformation framework supporting analysis of `HLCL` variability representations did not solve the interoperability issues. These results marked a change in the path to provide an expressive and flexible framework. From this point on, the concerns regarding variability modeling and variability analysis were treated independently in the thesis.

The next steps in the research pointed to answer the question *How intermediate representations can be integrated in a framework to ease the interoperability of variability management tools?*. The result is the proposal of Coffee's conceptual model including two intermediate languages, one for variability and one for analysis. Then, the framework solves the interoperability among modeling tools introducing an expressive textual variability modeling language called the High-Level Variability Language. Also, in Coffee the strong coupling and lack of flexibility in the transformation framework are solved by introducing a Generic Constraint Representation together with a context-aware transformation framework.

The High-Level Variability Language (HLVL) is an expressive and extensible textual language that can be used as a modeling and intermediate language for variability. HLVL was designed following an ontological approach, *i.e.,* by defining their elements considering the meaning of the existing concepts in different variability languages. This proposal not only provides a unified language based on a comprehensive analysis of the existing ones, but also sets foundations to build tools that support different notations and their combination in a concept-driven approach.

The evaluation of the framework comprised three stages. The first stage evaluated the ontological expressiveness of the HLVL. This evaluation recovered the theoretical framework developed in earlier stages of the research. The second stage demonstrates the feasibility of the proposal by implementing the framework and show its flexibility. Chapter 5 described how the context-aware transformation framework interacts with different set of rules and solvers. Also, this chapter presented the proposed architecture to implement this transformation framework together with a prototype implemented using Java, Xtext, and MiniZinc technologies. The final stage evaluates the applicability and usefulness of Coffee to solve the interoperability issues among different tools. This evaluation contrasts the framework proposed in this thesis against the requirements and scenarios for a unified feature notation defined by Berger & Collet [BC19] and discussed in the MODEVAR community.

6.2 Discussion and Limitations

To discuss the limitations of this proposal this section recalls the results reported in the dissertation.

6.2.1 About the Constraint-based Approaches for Variability Management

Chapter 2 presents a collection of concepts that contributes to describe variability-intensive systems more accurately. Nevertheless, to the best of our knowledge, there are not initiatives aiming to integrate all these

concepts in a unified language. One of the possible reasons is the trade-off between expressiveness and the analysis capabilities of the language. However, an unified variability modeling language may improve the expressiveness and accuracy of variability specifications.

The literature review evidenced the diversity of approaches to encode variability models as constraint satisfaction problems. The results showed that there are at least 23 different publications containing a set of transformation rules or an extension to a set of rules. One challenge in the research area is to consider this diversity and to develop an standard analysis library. The development of a standard analysis library shall propel the usage of variability management tools and unite the variability community efforts to provide a robust common analysis core.

There are two types of tools supporting constraint-based approaches: solvers and other software tools. Yet, there is no one solver specially designed for variability management. But, there is broad support for solving constraint problems. Also, there is no a particular solving paradigm for variability management. On the contrary, this mapping study showed that solvers in variability management range in a variety of paradigms, constraint systems and implementations. One challenge regarding solvers is to address the difficulty of exploiting the strengths of multiple solvers while solving a single complex problem.

6.2.2 HLVL as Modeling and Intermediate Language for Variability

One of the main characteristics of HLVL is that it contains constructs for comprehensively modeling variability concerning the concepts in the variability glossary presented in Section 2.3.1. Hence, HLVL can be used (1) as a specification language to create variability models; or (2) as an intermediate representation of models specified in other variability languages. Here, we borrow the concept of intermediate language from the compilers' domain. In this domain, intermediate languages are used to produce intermediate representations during the process of translating a source program into target code. Many compilers generate an explicit low-level or machine-like intermediate representation, which can be thought as a program for an abstract machine, as in the case of the Java language.

The usage of an intermediate language for variability is a viable alternative to the interoperability problem because written in such a language, variability models can be easily shared or distributed. Then, modeling tools should be able to export and import models in the intermediate language, so modelers do not have to learn a new variability language, then, modeling tools can be used as they are today.

An intermediate language for variability can be to variability modeling tools what the BibTeX format is for reference tools. That is, these managing references applications (e.g., Mendeley, Zotero, etc.) have their own formats and styles for managing references. Yet, these applications are also capable of importing and exporting BibTeX formats. Even electronic databases, for example, ACM data library, IEEE Xplore, Springer, Science Direct, citeSeer, etc. have their own way to store and display references. However, these electronic databases provide an option for downloading or exporting references in the BibTeX format. Moreover, in the rare cases a publication has not available a BibTeX format, it is possible to define it because the language' syntax is simple. Also, there exists examples and documentation for the BibTeX notations publicly available.

HLVL can be considered an exchange language for variability models considering the following factors.

- HLVL supports different modeling paradigms. As shown in Section 4.2.4 the language can be used at least to specify feature-oriented models (*e.g.,* FODA models, attribute-based feature models, cardinality-based feature models, variation-point oriented models (*e.g.,* OVM [PBvdL05], COVAMOF [SDNB04], CO-OVM [Dum14], CVL[Hau]) and decision-based model (*e.g.,* DoplerML). Moreover, the language provides numerous constructs to represent different types of variability units as options. Thus HLVL can represent unconventional variability languages.
- HLVL supports ad-hoc constraints. The literature review conducted in Chapter 2 showed that variability models are often enriched with ad-hoc constraints to gain expressiveness. These approaches contribute to the proliferation of new dialects and language extensions. Considering that the transformation of the base language into an HLVL model is viable, the new constraints can be directly written in HLVL and included in the model. Then, HLVL can be a standard language to add variability relations not supported by current notations to enhance variability models without increasing the variability of variability languages.
- HLVL's semantics is independent of any particular implementation. The language has a formal definition with a textual syntax using a free-context grammar and its operational semantics defined via a semantic domain, semantic function and inference rules. The formal definition of HLVL is key to avoid dependencies with any particular implementation and to ease the sharing making possible the implementation of *parsers* and code generators. Moreover, the formal definition of the syntax and semantics is a prerequisite for efficient and safe tool automation [SHTB07].
- As a textual variability language, HLVL deals with large models better than graphical languages. Graphical languages are not a viable option for large industrial-size variability models because they cannot be properly visualized. The high amount of elements in the model requires large physical space or dedicated tool support unless they are split in views. Additionally, most variability models have some textual elements such as cross-tree constraints, attributes and annotations that cannot be easily visualized because they clutter the layout. Consequently, textual variability languages, such as HLVL became popular because they are an alternative to solve the issues of large graphical models. Furthermore, what makes a graphical model more readable, *i.e.,* the layout and disposition of the objects in the model, is not conserved when sharing the model in different tools, because the interchange format is focused on variability and not graphical information.
- The concrete syntax of HLVL eases its usage as modeling language and exchange format. The language's concrete syntax resembles the syntax of declarative programming languages where elements and relations are the most important concepts. This programming-like syntax makes HLVL a human-readable language considering that nearly all computing professionals have come across a programming-like syntax and are thus familiar with this style. Moreover, HLVL is a lightweight language in contrast to the verbosity of other XML-based notations such as xmi, sxfmi and other syntaxes that even include information about graphical components in the variability models.
- The support of Boolean and non-Boolean variable items, the capability of modeling different variability languages, and the potential capability to enhance variability models suggest that HLVL is a viable language for integrating variability models described in different languages. Either written in the same editor or created in different modeling tools, models from different sources can be integrated to be analyzed or

configured.

6.2.3 About HLVL's Expressiveness

HLVL was designed following an ontological approach. On the one hand, the design of the language considered the results and lessons learned form the ontological evaluation presented in Chapter 3. On the other hand, the definition of the language considered the meaning of the concepts existing on different variability languages. That is, the glossary of variability modeling concepts surveyed in the literature review and reported in Chapter 2. Consequently, this proposal not only provides a unified language based on a comprehensive analysis of the existing ones, but also sets foundations to build tools that support different notations and their combination in a concept-driven approach. The following subsection discusses the results and limitations of HLVL regarding its expressiveness.

From the Ontological point of View

The results in the ontological analysis applied to HLVL under the Recker *et al.*'s metrics [RRIG09] showed that the degree of redundancy, the degree of overlapping and the degree of deficit, affect the ontological clarity and completeness of the language.

HLVL presents a medium level of clarity due to its levels of redundancy and overlap. The results in Chapter 5 showed that the degree of redundancy is 66% and the degree of overlapping is 8%. This redundancy and overlap levels depend on the number of constructs in HLVL. Then, the mapping between the language and the ontology is not one-to-one, it cannot be, since HLVL provides more constructs than the ontology. Additionally, the mapping shows that two or more constructs represent a single ontological construct. However, this redundancy and overlap in the language is intentional. Moreover, this redundancy is one of the strengths of the language. The multiple way to represent the variability concepts in HLVL aims to the flexibility of the language. Consequently, the language supports multiple modeling paradigms and allows the modeler to choose the way they feels more comfortable to define a variability model.

The decision to tolerate some redundancy and overlap in the language came from one of the conclusions in the ontological evaluation of the state-of-the-art `HLCL` framework. Then, HLVL contains a set of redundant constructs but also provides the mechanism to mitigate the impact of the redundancy in the clarity of the language. On the one hand, to enhance the clarity of the language and avoid questions such as, *Which construct should be used in a particular situation?*, HLVL provides constructs for frequently used variability relations and a classification of those constructs in the HLVL(x) sublanguages. Additionally, the design of the abstract libraries to map external languages to the expressiveness levels their contained sublanguages also provide a context to guide the modeler in the usage of the language.

HLVL presents a high level of ontological completeness as its degree of deficit is the 11%, however, the language is still incomplete. The ontological incompleteness exists because it is not possible to map any HLVL construct to the variability pattern of *ordering*. This conclusion is not surprising given that the logic representation used in the operational semantics of the language (e.g., first-order logic and concurrent constraint programming) does not support the concept of order or partial order in the resulting configurations. The inclusion of time as a

native concept in HLVL was considered in the design of the language for the following considerations. First, the results of the ontological evaluation of the the state-of-the-art `HLCL` framework, and the results of Asadi *et al.*'s study [AGWH12] concluded that variability modeling languages lack of completeness as they do not have any construct for representing computational time. Second, the notion of computational time, or order in the configuration seems relevant for some types of variability intensive systems such as, self-adaptive software systems, and dynamic software product lines. Third, a few works had considered the notion of computational time in variability modeling. The systematic literature review in Chapter 2 reported the existence of at least four works with these characteristics. These considerations raised the question of *should variability modeling languages include a construct representing computational time?*. This work explored this possibility by including visibility constructs, and conditioned implies and mutual exclusion. These constructs are transformed into *reified constraints* that trigger or hide some constraints in the resulting constraint problem. However, these constructs do not cause an order relation or partial order over the results of the configuration. To provide this order relations, the transformation framework must consider different logic representations and modifications to the solving algorithm as in the works of Sousa *et al.* [SRD17].

From the Comparison to the Glossary and other Textual Languages

The summary in Chapter 4 presented the Table 4.3 evaluating how HLVL supports the variability concepts in the glossary of concepts defined in the systematic literature review (*cf.* Chapter 2).

As shown in the table, as a whole, HLVL supports above the 90% of the variability modeling concepts in the glossary. Nevertheless, each HLVL(x) sublanguage has a different expressiveness level, and therefore, requires different support levels. The latter was considered in the design of the language's operational semantics. The remaining 10% of concepts not supported are related to the expressions language and its lack of support of first-order logic. The extension of the expressions language is an object of consideration as it may improve the expressiveness but also may impact the transformation framework in terms of the logic paradigms and solvers able to support such expressions.

Another consideration in the extension of the expressions language is to consider the inclusion of optimization expressions. An important number of works reported in the systematic literature reviews include constraints to include an objective function and optimize this function in the configuration. Again, this extension, as all the expressiveness decisions, may impact the transformation framework but is a more feasible extension since most solvers provide tools to implement optimization.

A final consideration is the result obtained to compare HLVL to other textual variability modeling languages under the categories in the classification of Eichelberger & Schmidt's and ter Beek *et al.* [ES15, tBSE19]. This comparison presented in Chapter 5, Table 5.8 confirms that HLVL lacks first-order logic support. Also, the table shows the lack of support for derived and user-defined data types and the definition of default values in the configuration. The current version of HLVL presented in this dissertation lacks scalability support via composition. However, the table shows an indirect support since there is an ongoing satellite project to provide this scalability support. The project is discussed in the future work below.

6.3 Future work

The following are ideas and prospects projects to further develop the proposal of this thesis. Also to pursue future research derived from the proposals in this thesis.

6.3.1 Extending Coffee

The future work regarding an in-deep evaluation and some extensions of the framework are part of an ongoing project derived from this research. The project was proposed and supervised by the first author of this thesis and funded by Universidad Icesi. The project main objective of this project is to develop a set of tools to support the integration of Coffee with other variability management tools. The project involves the participation of a group pf undergraduate students from the Software Engineering program at Universidad Icesi and two teachers (myself included). The activities in the project includes the implementation of the tools to Coffee to an early version of Variamos-Web using a microservices architecture. Also, the instantiation of the abstract encoding libraries to produce four parsers for VariaMos, FeatureIDE, Splot, and FAMILIAR.

The following activities considers the result obtained in the evaluation of the applicability and usefulness of Coffee by comparing the framework against the requirements and scenarios defined by Berger & Collet [BC19].

Scalability Support. Model composition is a key factor to provide a language that supports industrial-size models. As shown in the evaluation of Coffee with respect to the requirements and scenarios for a unified feature notation defined by Berger & Collet [BC19] HLVL does not support model composition, as it does not allow the specification or analysis of fractional models.

The extension to provide scalability support is the more advanced activity. The work related to extend the framework and provide model composition was conducted by Sara Drada and Juan Diego Carvajal. The students developed the idea to extend the framework as their final project under my supervision. The following are the results of the project.

- A Literature review to define six scenarios and requirements to support composition of variability modeling languages.
- The design and development of a prototype that implements the operations and techniques supporting composition.
- Extending HLVL to specify composed models.
- An evaluation of the extension surveying experts and comparing the correcteness of the results obtained by their prototype with respect to the results provided by FAMILIAR.

The reader may find the results associated to this project in the following repository: `https://github.com/coffeeframework/ModelOperations`.

Other extensions The following is a list of the most relevant extensions to bring Coffee closer to meet the scenarios and requirements to be considered as exchange framework [BC19].

- Implementation and testing of the libraries to import HLVL models in other tools. This item is relevant to completely meet the requirements in the *exchange scenario*.

- Extend the framework to support the mapping of options to implementation elements. This extension should explore the concept of abstract option, its syntax and semantics. The support of the link between options and implementation elements meet the requirements in the *mapping to implementation* and *model weaving scenarios*.
- Compatibility to other languages. All the implementation used java technologies. Though, the tools employed in the implementation, such as, Xtext to implement HLVL, MiniZinc as pivot language for logic representations, and the REST APIs provide flexibility in the interoperability to other tools and languages. However, more tests are required to support interactions with tools developed in different technologies. A first approximation in this regard is to implement an interpreter to execute MiniZinc specifications using GNUProlog.

6.3.2 Extending HLVL to Reach Ontological Completeness

The results of the ontological analysis of HLVL evidenced the lack of representation of one variability pattern: ordering. This pattern is related to the notion of time in variability management. However, it represents an order relation or a partial order between the elements in a configuration. The problem regarding the lack of mapping to the ordering pattern is that it reflects a limitation in the formalism used to provide its configurational semantics. Since, the concurrent constraint programming model does not support an order or partial order relation among the set of valuations solving a CCP [SR90].

Time is a notion that has been studied by people from different areas of knowledge. Depending on the point of view, it has a very unique definition, for instance, the physical time is the angle of the earth's rotation around its axis, the present time is the magnitude measured by clocks, and musical time is the measure of the beat, which is the basic rhythmic unit. However, although these definitions seem different, they all share the feel of movement and transition. This concept is usually represented in terms of temporal relations which are formalized with a symbolic logic.

As discussed in Chapter 3, time is a notion that could be useful to include in a variability modeling language as it is a relevant concept when modeling variability-intensive systems such as dynamic software product lines and self-adaptive software systems. A further step in the development of Coffee is to explore the usage of temporal logic [Pnu77] or the situation calculus [MH69] to provide variability relations such as the following:

- Configuration c_1 is valid in the situation where element x is *true*
- Configuration c_2 is no longer valid in the situation where element y is between 1 and 2.
- Configuration c_3 is valid in the situation where relation r_1 holds.

6.3.3 Research perspectives

This section presents a set of research perspectives and open open questions derived from the results and limitations of the research presented in this dissertation

- Many features of Coffee were designed following an ontological approach and developed from the perspective of Programming Languages Engineering. On the one hand, the approaches employed in the

exploration phase of the research are ontological. On the other hand, the results obtained using these ontological approaches guided most of the design phase. For instance, both, the systematic literature review and the ontological analysis are ontological approaches since the objective in both activities is to structure the knowledge using concepts and relationships among those concepts. Once the knowledge about variability modeling was structured in terms of concepts and relationships, the selection of a concept-driven approach, the introduction of the modeling paradigm, the definition of the incremental support levels, the sublanguages structure, and the mapping between the support levels, sublanguages, and other modeling languages came naturally. Mainly because of my academic and research background and because this research continues the idea that the inclusion of a pivot language enables the interchange of variability models originally specified in different modeling languages [DS08, SMD+11a, MSD11, MFTR+15]. An open question surging from pondering about what things could be done different is *What would be the results if the variability between modeling languages is addressed from a product line engineering approach?*

- The implementation of the context-aware transformation framework provided in this thesis relies on software variability. That is, the variability is supported using factories, abstract superclasses, interfaces facilitating different implementations, and conditional compilation (e.g., using #ifdefs) [MP14]. To recognize the variability in the domain and identify the variability in the implementation is one of the first steps in adopting a product line strategy. The next step is to explore the product line strategy or a dynamic software product line strategy to implement the transformation framework. Then, the question derived is *Should the context-aware transformation framework be designed as software product line or a Dynamic Software Product Line?* integrate into Coffee a DSPL approach may be convenient to provide the self-configuration capability to the framework.

- The idea to provide an intermediate variability language to solve the interchange issues among tools has been addressed by other authors. More recently, similar proposals arose to the ones presented in this thesis. The ideas developed jointly by Sundermann *et al.*[SFE+21], and Feichtinger *et al.*[FR21] propose a variability modeling language called UVL aiming to become a standard format for exchange between variability modeling tools. Also, to provide a flexible transformation framework to avoid the strong coupling between the logic paradigm, transformation rules, and solving mechanisms. These similar approaches confirm the relevance of the problematic developed in this thesis. Thus, far from being outdated, modeling variability and analyzing variability models still pose challenges to the variability management community. In this regard the following questions arise,

  - UVL is a feature-oriented language and HLVL is a multiparadigm modeling language, *what are the commonalities among both languages?*, are they dissimilar? are they complimentary?

  - UVL's syntax was designed evaluating the preferences of the community regarding the language's scope and its syntax, *How close/far is HLVL from these community preferences? which preferences can be incorporated into HLVL?*, *Does the inclusion of the community preferences in HLVL affect its multiparadigm capabilities?*

  - *Can the theoretical evaluation framework proposed in this thesis be applied to evaluate UVL's ontological expressiveness? ¿Are the results comparable to the ones obtained for HLVL?*

Part IV

Appendix

Appendix A

Systematic Mapping Study: Protocol and Artifacts

A.1 Search Terms

Table A.1: Search terms

Set 1: Variability management terms
Product line, PL, software product line, SPL, dynamic software product line, DSPL, product family, feature oriented, feature model, feature modeling, variability, variation model, variability model, variability modeling, reuse management, software reuse
Set 2: Constraint-based terms
Constraint satisfaction problem, CSP, constraint programming, solver, dynamic constraint optimization problem, DCOP, dynamic CSP, flexible CSP, weighted CSP, fuzzy CSP, Distributed CSP, Constraint logic programming, CLP, logic programming, logic formula, SAT solver, SMT solver, BDD solver,
Set 3: Solver-related terms
Absolver, AIMMS, Alma-0, Alt-Ergo, AMPL, Artelys Kalis, Babelsberg, Barcelogic, BASOLVER, Beaver, Boolector, Cassowary, Cbc, chip V5, choco, Ciao, Claire, Comet, Cream, Curry, CVC3, CVC4, DBPLL, Disolver, DPT, Emma, ECLiPSe, FlatZinc, F1 Compiler, Gecode, GLOP, Glpk, Glucose, Google CP Solver, Gurobi, ILOG CPLEX, JaCoP, Jaopt, Jekejeke, JSR- 331, Koalog Constraint Solver, Lingeling, Logic-tools, Minion, Minlog, Minizinc, Numberjack, OptaPlanner, OpenCog, OscaR, Oz, Picat, prolog, python constraint, SCIP, Screamer, SMOCS, SMCHR, SONOLAR, somerby web solver, Spear, STP, Sulum, SWORD, Turtle, UCLID, verilog, veriT, Yices, zChaff, Z3

A.2 List of selected venues for manual search

Table A.2: Selected venues

Conferences and workshops
Automated Software Engineering (ASE), International Conference on Software Engineering (ICSE), International Conference on Progress in Informatics and Computing (PIC), International Conference on Software Testing, Verification, and Validation (ICST), W para workshops International Systems and Software Product Lines Conference (SPLC), International Conference on Tools with Artificial Intelligence (ICTAI), International Workshop on Product Line Approaches in Software Engineering ICSE (PLEASE), International Workshop on Variability Modelling of Software-intensive Systems (VaMos)

Journals

Automated Software Engineering Journal (ASE), Constraints an International Journal (Constraints), The CP Journal (CP), Empirical Software Engineering (ESE), Information and Software Technology (IST), Journal of Systems and Software (JSS)

A.3 Data Extraction Instruments

A.3.1 Data Extraction Process

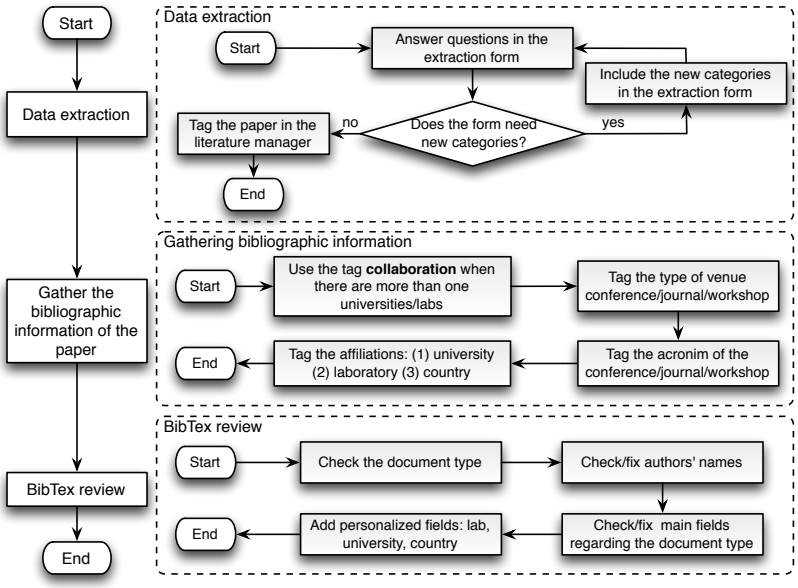


Figure A.1: Data extraction process

A.3.2 Data Extraction Questionnaire

DE-Q1 What is/are the paradigm(s) to describe variability?

DE-Q2 What are the variability's concepts used by the variability notation?

DE-Q3 Does the publication includes new constraints (different from the provides by the variability notation)? What is the purpose of the inclusion of new constraints?

DE-Q4 Which rules were applied to transform variability models into constraints?

DE-Q5 What are the types of the constraints and domains used in the transformation?

DE-Q6 Does the publication reports the usage of a solver?, Which solver?, What is the solver's paradigm?, How is the solver implemented? (as a library, language, stand-alone tool)

DE-Q7 Does the publication relates a constraint-based software-tool, or presents the implementation of a software-tool?, What is the name of the tool?, What is the type of tool?, What is the software tool used for?

A.4 Bibliometric Information

A.4.1 Bibliographic questions

1. What is the name and the acronym of the venue?
2. What is the type of the venue?
3. What are the filliation of the authors?
4. Where are the laboratories/universities from?
5. What is the category in the Wieringa et al. [WMMR05] classification?
6. Based in the guidelines by Petersen et al. [PVK15], what is the the evidence level in a the research paper?
7. What type(s) of product family(ies) is/are targeted in the publication?
8. What is/are the type(s) of the contribution(s) of the publication?
9. What is/are the SPLE task(s) addressed by the publication?
10. What is/are the SPLE process(es) [PBvdL05] addressed by the publication?

A.4.2 Results relevant authors and fora

RQ1 aims to procure a better understanding of the publication information of publications in the area by finding the most relevant authors, papers, and venues related to the topic of constraint-based approaches on SPLE. First, this section addresses the fora for publications on the topic of CP in the context of SPLE. To this purpose, the process for reviewing each document included a step for gathering the bibliographic information of each paper such as the type and name of the venue, the acronym, and the source of retrieval. Fig. A.3 presents the obtained results. In consequence, the collection of 137 selected papers contains 101 documents classified as proceedings (79 from conferences and 22 from workshops), 26 journal articles, and ten book chapters. Fig. A.3 also depicts the distribution of publications over types of venues such as journals, conferences, workshops, and book chapters. For instance, in the bar for Conferences, the most frequent conference is SPLC with 24 publications, followed by ICSE with five publications. In addition, in the bar for Journals, the most frequent journal is Systems and Software (S&S) with four publications. There is a tie for the second most frequent journal between the Science of Computer Journal, the Software Quality Journal, and the Information and Software Technology Journal with two publications each. Additionally, in the Workshop bar, the most frequent workshop is VaMos with fourteen publications. Finally, the distribution of the publications in the bar book-chapter is uniform. Table A.3 contains a summary of the information of the most relevant venues: type of venue, acronym, venue's name, the source of retrieval, and the amount of selected papers.

Most relevant authors and publications

To find the most relevant publications in the research area, we used the tools provided by the reference manager to provide the list of publications cited in each paper that are also part of the selected papers collection. From

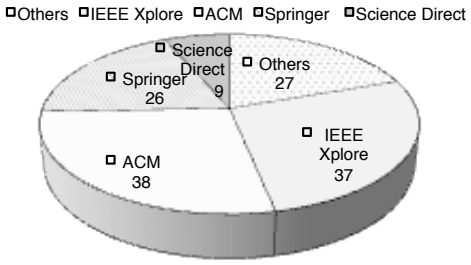


Figure A.2: Distribution of documents retrieved per source.

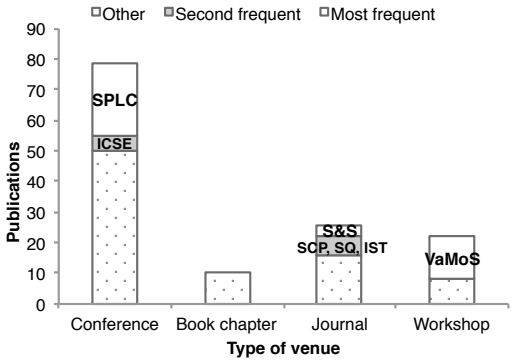


Figure A.3: Amount of publications per type of venue.

this list and with the help of a Java program we obtain the number of citations of each paper. As a result, the "Feature models, grammars, and propositional formulas" paper [Bat05] is the most cited one with 58 studies referencing Batory's proposal for representing feature models as formulas in propositional logic. The second and third most cited papers are the proposals for transforming extended feature models into constraint satisfaction problems presented by [BTRC05a] and [CK05]. Fig. A.5 shows the twenty most cited papers over the 137 selected documents in this mapping study.

Fig. A.4 depicts the 20 most influential authors regarding the proposed indicators. This figure presents four indicators used to evaluate the impact of authors in the research area. These indicators are defined as follows:
Author/Co-author = *total of publications as author.*
First Author = *total of publications as first author.*
Citations ratio = *the sum of citations of each paper where the author is first contributor.*
Citations index = *Citations ratio / first author.*

To illustrate, consider the indicators for D. Benavides, first author in Fig. A.4. The first bar represents the total publications where D. Benavides is cited as one of the authors (16 papers). Whereas, the second bar is the number of documents where D. Benavides is the first author (5 papers). The third bar is the sum of all citations for the publications where D. Benavides is the first author (citations ratio = 82), and the fourth bar is the citation index (citations ratio/ first author = 82/ 5 = 16,4). Note that the three first authors in Fig. A.4 correspond to the principal contributors of the most cited papers as presented in Fig. A.5

Table A.3: Most frequent venues

Type	Acronym	Name	Source	Papers
Conference	SPLC	International Software Product Line Conference	ACM-DL, IEEE Xplore	24
Conference	ICSE	International Conference on Software Engineering	IEEE Xplore	5
Conference	RE	Requirements Engineering Conference	IEEE Xplore	4
Conference	CAiSE		Springer	4
Journal	S&S	System and Software	Science Direct	4
Journal	SCP	Science of Computer Programming	Science Direct	2
Journal	SQ	Software Quality Journal	Springer	2
Journal	SIT	Information and Software Technology	Science Direct	2
Workshop	VaMoS	International Workshop on Variability Modelling of Software-intensive Systems	ACM-DL	14

A.4.3 Results for types of research and evaluation

One of the objectives of this mapping study is the characterization of the type of research together with the evaluation methods adopted in primary studies to delineate the maturity of research in the area. To address RQ2, we employed two complementary points of view: (1) Wieringa et al.'s scheme to classify the type of research [WMMR05], and (2) the guidelines provided by Petersen et al. in [PVK15] to evaluate the evidence level in a research paper. The categories in the the Wieringa et al.'s classification and the type of evaluation were included in the classification framework and described in Section **??**. Table A.4 presents summary of the categories.

Fig. A.6 contains the distribution of papers regarding the Wieringa et al.'s scheme. One of the results is that there are not publications in the category experience paper. This phenomenon is a consequence of the exclusion of secondary studies from the final collection of documents along with the distribution of the type of research in the area. As shown in Fig. A.6, the majority of studies are in the solution proposal category (66%), meanwhile, 31 publications are validation research studies (23%), ten publications are philosophical papers (7%) and only six studies are in the research evaluation category (4%). These numbers show that in spite of the many solution proposal and validation research studies, there is still an incipient amount of research reporting lessons learned and practical applications. Therefore the lack of experience papers.

Fig. A.7 displays the results of the evidence provided from primary studies. Accordingly with this chart, the main strategies for empirical evaluation are academic case studies (51%), and academic (big-set) (15%). On contrast, the less used strategies for empirical evaluation are industrial cases (13%), and industrial practice (4%). It is also observe form Fig. A.7, that the 12% of studies not provide empirical evaluation and 6% used toy-examples to evaluate their proposals. Further development of the evaluation of constraint-based approaches for SPLE on industrial contexts and industrial practice is required to provide better evidence about the scalability

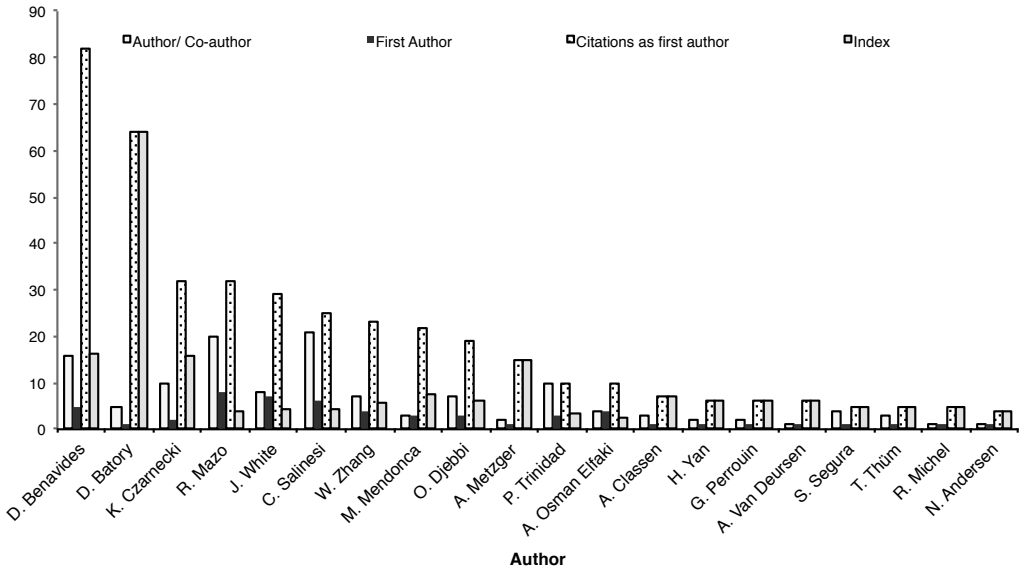


Figure A.4: Relevant authors in the research area.

and suitability of constraint-based approaches in SPLE.

To provide a better understanding and conclude about the research maturity of CP approaches applied to SPLE, Fig. A.8 displays the type of research and evidence level with respect to the target product line. In this figure, the size of the bubble represents the number of publications. Therefore, Fig. A.8 shows that most publications aim to employ CP approaches in software product lines than in product lines. In addition, despite there are some works involving dynamic software product lines; there is still a gap in the research concerning constraint-based techniques applied to that type of software product lines

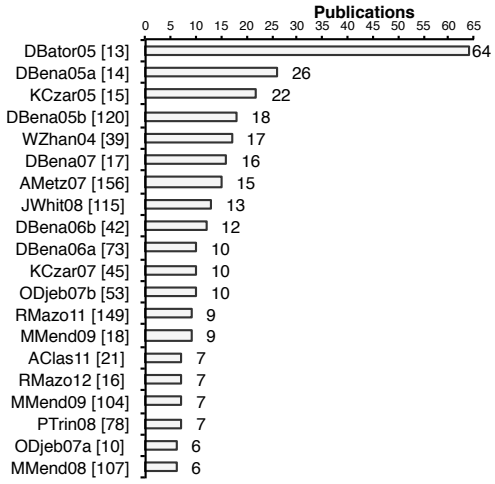


Figure A.5: Most cited papers.

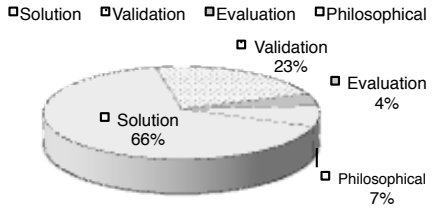


Figure A.6: Distribution of the publications per type of research.

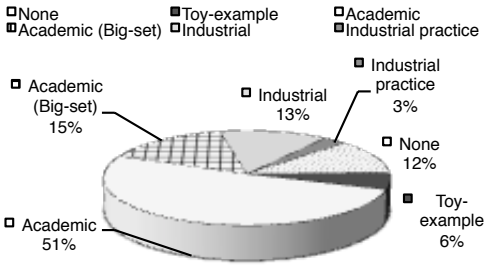


Figure A.7: Distributions of documents per level of evidence.

Table A.4: Summary of the evaluation categories

Type of research (Wieringa et al. [WMMR05])
Philosophical paper. Papers that sketch a new way of looking at things, a new conceptual framework.
Experience papers. Papers explaining how something has been done in practice.
Solution proposal. Papers discussing new or revised techniques.
Validation research. Papers discussing the properties of solution proposals not yet implemented in practice.
Evaluation research. Papers with techniques implemented in practice and reporting the learned lessons.
Evaluation level (Petersen et al. [PVK15])
None. No empirical evidence, the evidence may be provided from observations, demonstration or arguments.
Toy-example. The evidence is obtained by demonstration on toy-examples.
Academic. The evidence is obtained by working out on case studies employed on other publications or repositories.
Academic (Big-set). Evidence obtained using case studies reported as real life size, or big. These cases are randomly generated or obtained from public repositories.
Industrial. Evidence obtained using case studies inspired from industrial partners.
Industrial practice. The contributions of he publication is used in an industrial context.

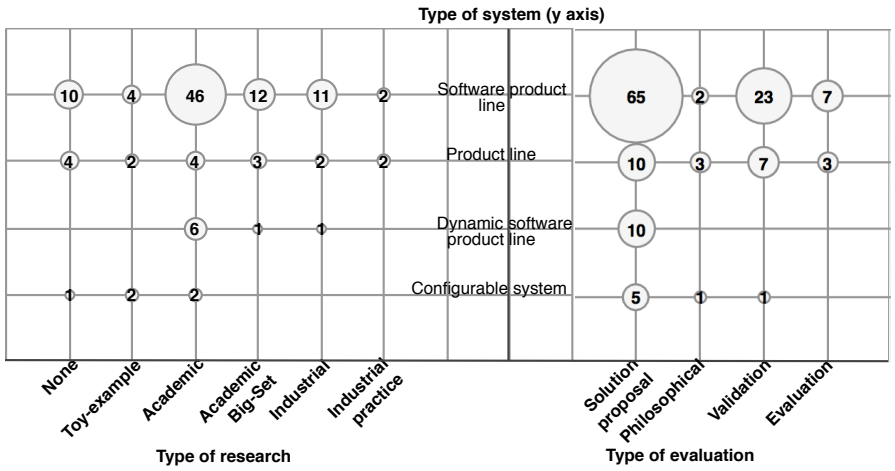


Figure A.8: Research type and evaluation level regarding the type of system.

References

[ACF10] Jean-Marc Astesana, Laurent Cosserat, and Helene Fargier. Constraint-based Vehicle Configuration: A Case Study. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 68–75. IEEE, IEEE, oct 2010.

[ACF14] Juliana Alves Pereira, Kattiana Constantino, and Eduardo Figueiredo. A Systematic Literature Review of Software Product Line Management Tools. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8919:73–89, 2014.

[ACL13] César Andrés, Carlos Camacho, and Luis Llana. A formal framework for software product lines. *Information and Software Technology*, 55(11):1925–1947, 2013.

[ACLF11] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B France. Managing feature models with familiar: a demonstration of the language and its tool support. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, pages 91–96. ACM, 2011.

[ACLF13] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming*, 78(6):657 – 681, 2013.

[ACSW12] Nele Andersen, Krzysztof Czarnecki, Steven She, and Andrzej W\kasowski. Efficient synthesis of feature models. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*, pages 106–115. ACM, 2012.

[ADD+00] Mark Ardis, Peter Dudak, Liz Dor, Wen-jenq Leu, Lloyd Nakatani, Bob Olsen, and Paul Pontrelli. Domain engineered configuration control. In *Software Product Lines*, pages 479–493. Springer, 2000.

[AGV15] Paolo Arcaini, Angelo Gargantini, and Paolo Vavassori. Generating Tests for Detecting Faults in Feature Models. In *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*, pages 1–10. IEEE, 2015.

[AGWH12] Mohsen Asadi, Dragan Gasevic, Yair Wand, and Marek Hatala. Deriving variability patterns in software product lines by ontological considerations. In *Conceptual Modeling – ER*, volume 7532 LNCS, pages 397–408, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[APM+14] Germán Harvey Alférez, Vicente Pelechano, Raúl Mazo, Camille Salinesi, and Daniel Diaz. Dynamic adaptation of service compositions with variability models. *Journal of Systems and Software*, 91:24–47, may 2014.

[ApsGtFIfOCSF] Daimler AG, pure-systems GmbH, and the Fraunhofer Institute for Open Communication Systems FOKUS. The variability exchange language (draft version).

[Apt03] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[ARS+19] Asmaa Achtaich, Ounsa Roudies, Nissrine Souissi, Camille Salinesi, and Raúl Mazo. Evaluation of the state-constraint transition modelling language: A goal question metric approach. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B*, SPLC '19, page 106–113, New York, NY, USA, 2019. Association for Computing Machinery.

[AS96] Harold Abelson and Gerald J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, USA, 2nd edition, 1996.

[ASN14] Stephan Adelsberger, Stefan Sobernig, and Gustaf Neumann. Towards assessing the complexity of object migration in dynamic, feature-oriented software product lines. In *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*, page 17. ACM, 2014.

[ASS+19] Asmaa Achtaich, Nissrine Souissi, Camille Salinesi, Raul Mazo, and Ounsa Roudies. A constraint-based approach to deal with self-adaptation: The case of smart irrigation systems. *International Journal of Advanced Computer Science and Applications*, 10(7), 2019.

[Bat05] Don Batory. Feature models, grammars, and propositional formulas. In Henk Obbink and Klaus Pohl, editors, *Software Product Lines*, pages 7–20, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BC05] Felix Bachmann and Paul Clements. Variability in software product lines. Technical Report CMU/SEI-2005-TR-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.

[BC19] Thorsten Berger and Philippe Collet. Usage scenarios for a common feature modeling language. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B*, SPLC ?19, page 174?181, New York, NY, USA, 2019. Association for Computing Machinery.

[BCR94] Victor R Basili, Gianluigi Caldiera, and H Dieter Rombach. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[BDA+13] Kacper Bąk, Zinovy Diskin, Michał Antkiewicz, Krzysztof Czarnecki, and Andrzej Wąsowski. Clafer: unifying class and feature modeling. *Software & Systems Modeling*, pages 1–35, 2013.

[BDRG10] Ebrahim Bagheri, Tommaso Di Noia, Azzurra Ragone, and Dragan Gasevic. Configuring software product line feature models based on stakeholders' soft and hard requirements. In *Software Product Lines: Going Beyond*, pages 16–31. Springer, 2010.

[Ben19] David Benavides. Variability modelling and analysis during 30 years. In Maurice H. ter Beek, Alessandro Fantechi, and Laura Semini, editors, *From Software Engineering to Formal*

Methods and Tools, and Back: Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday, pages 365–373. Springer International Publishing, Cham, 2019.

[Ber12] Thorsten Berger. *Variability Modeling in the Real An Empirical Journey from Software Product Lines to Software Ecosystems*. PhD thesis, University of Leipzig, 2012.

[BHST04] Yves Bontemps, Patrick Heymans, Pierre-Yves Schobbens, and Jean-Christophe Trigaux. Semantics of FODA Feature Diagrams. In *Workshop on Software Variability Management for Product Derivation*, pages 48–58, 2004.

[BJM08] Andrew Burton-Jones and Peter Meso. The Effects of Decomposition Quality and Multiple Forms of Information on Novices' Understanding of a Domain from a Conceptual Model. *Journal of the Association for Information Systems*, 9(12):1, 2008.

[BLL+13] Johannes Bürdek, Sascha Lity, Malte Lochau, Markus Berens, Ursula Goltz, and Andy Schürr. Staged configuration of dynamic software product lines with complex binding time constraints. In *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems - VaMoS '14*, pages 1–8, New York, New York, USA, 2013. ACM Press.

[BNB14] Razieh Behjati, Shiva Nejati, and Lionel C Briand. Architecture-level configuration of large-scale embedded software systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(3):25, 2014.

[BSL+13] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki. A study of variability models and languages in the systems software domain. *IEEE Transactions on Software Engineering*, 39:1611–1640, 2013.

[BSRC09] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models: A detailed literature review. Research Report ISA-09-TR-04, Applied Software Engineering Research Group, December 2009.

[BSRC10] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, sep 2010.

[BSTRC06a] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. A first step towards a framework for the automated analysis of feature models. *Proc. Managing Variability for Software Product Lines: Working With Variability Mechanisms*, pages 39–47, 2006.

[BSTRC06b] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. Using java csp solvers in the automated analyses of feature models. In Ralf Lämmel, João Saraiva, and Joost Visser, editors, *Generative and Transformational Techniques in Software Engineering: International Summer School, GTTSE 2005, Braga, Portugal, July 4-8, 2005. Revised Papers*, pages 399–408. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[BSTRC07] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. FAMA: Tooling a Framework for the Automated Analysis of Feature Models. *VaMoS*, 2007:1, 2007.

[BTCS13] David Benavides, Pablo Trinidad, Antonio Ruiz Cortés, and Sergio Segura. Fama. In *Systems and Software Variability Management, Concepts, Tools and Experiences*, pages 163–171. 2013.

[BTRC05a] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Automated Reasoning on Feature Models. In *Advanced Information Systems Engineering*, pages 491–503. Springer, 2005.

[BTRC05b] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Using Constraint Programming to Reason on Feature Models. In *SEKE*, pages 677–682, 2005.

[CBH11] Andreas Classen, Quentin Boucher, and Patrick Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Science of Computer Programming*, 76:1130 – 1143, 2011.

[CGR+12] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, pages 173–182, NY, USA, 2012. ACM.

[CHE05] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, jan 2005.

[CHH09] Andreas Classen, Arnaud Hubaux, and Patrick Heymans. A Formal Semantics for Multi-level Staged Configuration. *VaMoS*, 9:51–60, 2009.

[CK05] Krzysztof Czarnecki and Chang Hwan Peter Kim. Cardinality-based feature modeling and constraints: A progress report. In *International Workshop on Software Factories*, pages 16–20, 2005.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[CN01] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, 2001.

[CW07] Krzysztof Czarnecki and Andrzej Wasowski. Feature diagrams and logics: There and back again. In *Proceedings of the 11th International Software Product Line Conference*, SPLC '07, pages 23–34, Washington, DC, USA, 2007. IEEE Computer Society.

[CWD+14] Maxime Cordy, Marco Willemart, Bruno Dawagne, Patrick Heymans, and Pierre-Yves Schobbens. An Extensible Platform for Product-line Behavioural Analysis. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, SPLC '14, pages 102–109, New York, NY, USA, 2014. ACM.

[DBN16] Alexander Diedrich, Björn Böttcher, and Oliver Niggemann. Exposing design mistakes during requirements engineering by solving constraint satisfaction problems to obtain minimum correction subsets. In van den Herik J Filipe J. Filipe J., editor, *ICAART 2016 - Proceedings*

of the 8th International Conference on Agents and Artificial Intelligence, volume 2, pages 280–287. SciTePress, 2016.

[DBS+17] Amador Durán, David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. FLAME: a formal framework for the automated analysis of software product lines validated by automated specification testing. *Software & Systems Modeling*, 16(4):1049–1082, oct 2017.

[DFH11] Deepak Dhungana, Andreas Falkner, and Alois Haselböck. Configuration of cardinality-based feature models using generative constraint satisfaction. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 100–103. IEEE, 2011.

[DGR11] Deepak Dhungana, Paul Grünbacher, and Rick Rabiser. The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering*, 18(1):77–114, mar 2011.

[DHR10] Deepak Dhungana, Patrick Heymans, and Rick Rabiser. A formal semantics for decision-oriented variability modeling with DOPLER. In *Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria, January 27-29, 2010. Proceedings*, pages 29–35, 2010.

[DKL+16] Frederik Deckwerth, Géza Kulcsár, Malte Lochau, Gergely Varró, and Andy Schürr. Conflict detection for edits on extended feature models using symbolic graph transformation. In Thum T Rubin J., editor, *Electronic Proceedings in Theoretical Computer Science, EPTCS*, volume 206, pages 17–31. Open Publishing Association, 2016.

[DMSEB15] Lamiae Dounas, Raúl Mazo, Camille Salinesi, and Omar El Beqqali. Continuous monitoring of adaptive e-learning systems requirements. In *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*, volume 2016-July. IEEE Computer Society, 2015.

[DS00] Bryan S Doerr and David C Sharp. Freeing product line architectures from execution dependencies. In *Software Product Lines*, pages 313–329. Springer, 2000.

[DS07] Olfa Djebbi and Camille Salinesi. Red-pl, a method for deriving product requirements from a product line requirements model. In John Krogstie, Andreas Opdahl, and Guttorm Sindre, editors, *Advanced Information Systems Engineering: 19th International Conference, CAiSE 2007, Trondheim, Norway, June 11-15, 2007. Proceedings*, pages 279–293, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[DS08] Olfa Djebbi and Camille Salinesi. Towards an Automatic PL Requirements Configuration through Constraints Reasoning. In *Second International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, pages 17–23, 2008.

[DSD07] Olfa Djebbi, Camille Salinesi, and Daniel Diaz. Deriving product line requirements: the red-pl guidance approach. In *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, pages 494–501. IEEE, 2007.

[DTS+14] Cosmin Dumitrescu, Patrick Tessier, Camille Salinesi, Sebastien Gérard, Alain Dauron, and Raul Mazo. Capturing Variability in Model Based Systems Engineering. In *Complex Systems Design & Management*, pages 125–139. Springer International Publishing, 2014.

[Dum14] Cosmin Dumitrescu. *CO-OVM : A Practical Approach to Systems Engineering Variability Modeling*. PhD thesis, Universit{é} Paris 1 Panth{é}on-Sorbonne, 2014.

[EFV+13] Osman Abdelrahman Elfaki, Sim Liew Fong, P Vijayaprasad, Md Gapar Md Johar, and Murad Saadi Fadhil. Using a Rule-based Method for Detecting Anomalies in Software Product Line. *Research Journal of Applied Sciences*, 2013.

[EKS13] Holger Eichelberger, Christian Kröher, and Klaus Schmid. An analysis of variability modeling concepts: Expressiveness vs. analyzability. In John Favaro and Maurizio Morisio, editors, *Safe and Secure Software Reuse*, pages 32–48, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Eng20] Dominik Engelhardt. *Towards a Universial Variability Language*. PhD thesis, Technische Universitat Braunschweig, 2020.

[EPAH08] Abdelrahman Osman Elfaki, Somnuk Phon-Amnuaisuk, and Chin Kuan Ho. Knowledge Based Method to Validate Feature Models. In *SPLC (2)*, pages 217–225, 2008.

[EPAH09a] Abdelrahman Osman Elfaki, Somnuk Phon-Amnuaisuk, and Chin Kuan Ho. Modeling variability in software product line using first order logic. In *International Conference on Software Engineering Research, Management and Applications, SERA 2009*, pages 227–233. IEEE, 2009.

[EPAH09b] Abdelrahman Osman Elfaki, Somnuk Phon-Amnuaisuk, and Chin Kuan Ho. Using First Order Logic to Validate Feature Model. In *VaMoS*, pages 169–172, 2009.

[ES15] Holger Eichelberger and Klaus Schmid. Mapping the design-space of textual variability modeling languages: A refined analysis. *Int. J. Softw. Tools Technol. Transf.*, 17(5):559–584, October 2015.

[FBGR13] A Felfernig, D Benavides, J Galindo, and F Reinfrank. Towards Anomaly Explanation in Feature Models. In *Workshop on Configuration*, pages 117–124, 2013.

[FO09] Raphael Finkel and Barry O'Sullivan. Reasoning about conditional constraint specifications. In *Tools with Artificial Intelligence, 2009. ICTAI'09. 21st International Conference on*, pages 349–353. IEEE, 2009.

[FR21] Kevin Feichtinger and Rick Rabiser. *How Flexible Must a Transformation Approach for Variability Models and Custom Variability Representations Be?*, page 69–72. Association for Computing Machinery, New York, NY, USA, 2021.

[GAT+16] José A. Galindo, Mathieu Acher, Juan Manuel Tirado, Cristian Vidal, Benoit Baudry, and David Benavides. Exploiting the enumeration of all feature model configurations. In *Proceedings of the 20th International Systems and Software Product Line Conference*, Splc'16, pages 74–78, 2016.

[GBT+18] José A. Galindo, David Benavides, Pablo Trinidad, Antonio-Manuel Gutiérrez-Fernández, and Antonio Ruiz-Cortés. Automated analysis of feature models: Quo vadis? *Computing*, 49(12):45, aug 2018.

[GDR+15] José A. Galindo, Deepak Dhungana, Rick Rabiser, David Benavides, Goetz Botterweck, and Paul Grünbacher. Supporting distributed product configuration by integrating heterogeneous variability modeling approaches. *Information and Software Technology*, 62:78 – 100, 2015.

[GFG08] Giancarlo Guizzardi, Ricardo Falbo, and Renata Guizzardi. Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology. In *Memorias de la XI Conferencia Iberoamericana de Software Engineering (CIbSE 2008), Recife, Pernambuco, Brasil, February 13-17, 2008*, pages 127—-140, 2008.

[GFG12] Renata Guizzardi, Xavier Franch, and Giancarlo Guizzardi. Applying a foundational ontology to analyze means-end links in the $i^\star$ framework. In *2012 Sixth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–11. IEEE, may 2012.

[GMS15] Arnaud Gotlieb, Dusica Marijan, and Sagar Sen. Towards more relational feature models. In Maciaszek L Lorenz P. Maciaszek L., editor, *ICSOFT-EA 2015 - 10th International Conference on Software Engineering and Applications, Proceedings; Part of 10th International Joint Conference on Software Technologies, ICSOFT 2015*, pages 381–386. SciTePress, 2015.

[GRF+12] Nadia Gamez, Daniel Romero, Lidia Fuentes, Romain Rouvoy, and Laurence Duchien. Constraint-based self-adaptation of wireless sensor networks. In *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups*, pages 20–27. ACM, 2012.

[GTBW14] José A Galindo, Hamilton Turner, David Benavides, and Jules White. Testing variability-intensive systems using automated analysis: an application to Android. *Software Quality Journal*, pages 1–41, 2014.

[Gui13] Giancarlo Guizzardi. Ontology-Based Evaluation and Design of Visual Conceptual Modeling Languages. In Iris Reinhartz-Berger, Arnon Sturm, Tony Clark, Sholom Cohen, and Jorn Bettin, editors, *Domain Engineering*, pages 317–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[GWT+14] Matthias Galster, Danny Weyns, Dan Tofan, Bartosz Michalik, and Paris Avgeriou. Variability in Software Systems—A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 40(3):282–306, mar 2014.

[Hau] Øystein Haugen. Common variability language (cvl) - omg revised submission.

[HBG11] Aymeric Hervieu, Benoit Baudry, and Arnaud Gotlieb. Pacogen: Automatic generation of pairwise test configurations from feature models. In *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on*, pages 120–129. IEEE, 2011.

[Hei13] Richard Heijblom. Potential of Integer Programming for Optimization Analysis of Extended Feature Models. 2013.

[Hev07] Alan R Hevner. A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, 19(2):87–92, 2007.

[HMGB16] Aymeric Hervieu, Dusica Marijan, Arnaud Gotlieb, and Benoit Baudry. Practical minimization of pairwise-covering test configurations using constraint programming. *Information and Software Technology*, 71:129–146, 2016.

[HMPR04] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Q.*, 28(1):75–105, 2004.

[HPF20a] Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes. Extensible and modular abstract syntax for feature modeling based on language constructs. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*, SPLC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[HPF20b] Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes. Extensible and modular abstract syntax for feature modeling based on language constructs. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*, SPLC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[HPHLT15] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proceedings - International Conference on Software Engineering*, volume 1, pages 517–528. IEEE Computer Society, 2015.

[HPP+13] Christopher Henard, Mike Papadakis, Gilles Perrouin, John Klein, and Yves Le Traon. Towards automated testing and fixing of re-engineered feature models. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 1245–1248. IEEE, 2013.

[HR04] D. Harel and B. Rumpe. Meaningful modeling: what's the semantics of "semantics"? *Computer*, 37(10):64–72, oct 2004.

[Iiv89] Juhani Iivari. Levels of abstraction as a conceptual framework for an information system. In Eckhard D. Falkenberg and Paul Lindgreen, editors, *Information System Concepts: An in Depth Analysis : Proceedings of the Ifip Tc 8/Wg 8.1 Working Conference on Information System Concepts : An in De*, pages 323–352, 1989.

[JBMS14] Mikoláš Janota, Goetz Botterweck, and Joao Marques-Silva. On lazy and eager interactive reconfiguration. In *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*, page 8. ACM, 2014.

[KAT16] Matthias Kowal, Sofia Ananieva, and Thomas Thüm. Explaining anomalies in feature models. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative*

Programming: Concepts and Experiences - GPCE 2016, pages 132–143, New York, New York, USA, 2016. ACM Press.

[KCH+90] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Software Engineering Institute, Carnegie Mellon University, 1990.

[KOD10a] A. Karatas, H. Oguztüzün, and A.H. Dogru. Global constraints on feature models. In *16th International Conference, CP. Proceedings*, pages 537–551. 2010.

[KOD10b] Ahmet Serkan Karataş, Halit Oğuztüzün, and Ali Doğru. Mapping extended feature models to constraint logic programming over finite domains. In Jan Bosch and Jaejoon Lee, editors, *Software Product Lines: Going Beyond*, pages 286–299, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[KOD13] Ahmet Serkan Karataş, Halit Oğuztüzün, and Ali Doğru. From extended feature models to constraint logic programming. *Science of Computer Programming*, 78(12):2295–2312, dec 2013.

[KPP+02] Barbara Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, aug 2002.

[Kru07] Charles W. Krueger. Biglever software gears and the 3-tiered spl methodology. In *Companion to the 22Nd ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion*, OOPSLA '07, pages 844–845, New York, USA, 2007. ACM.

[KTS16] Sebastian Krieter, Thomas Thüm, and Gunter Saake. Comparing algorithms for efficient feature-model slicing. In *Proceedings of the 20th International Systems and Software Product Line Conference*, pages 60–64, New York, NY, USA, 2016. ACM.

[Läm18] Ralf Lämmel. *Software Languages, Syntax, Semantics, and Metaprogramming*. Springer International Publishing, Cham, 2018.

[LC13] Miguel A Laguna and Yania Crespo. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming*, 78(8):1010–1034, 2013.

[LDSSH15] Patrick Leserf, Pierre De Saqui-Sannes, and Jérôme Hugues. Multi domain optimization with sysml modeling. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, volume 2015-Octob. Institute of Electrical and Electronics Engineers Inc., 2015.

[LDSSHC15a] Patrick Leserf, Pierre De Saqui-Sannes, Jérôme Hugues, and Khaled Chaaban. *Architecture optimization with sysML modeling: A case study using variability*, volume 580, pages 311–327. Springer International Publishing, 2015.

[LDSSHC15b] Patrick Leserf, Pierre De Saqui-Sannes, Jérôme Hugues, and Khaled Chaaban. Sysml modeling for embedded systems design optimization: A case study. In Desfray P Filipe J Filipe J Hammoudi S. Pires L.F., editor, *MODELSWARD 2015 - 3rd International Conference on Model-Driven Engineering and Software Development, Proceedings*, pages 449–457. SciTePress, 2015.

[LGCR15] Jia Hui Liang, Vijay Ganesh, Krzysztof Czarnecki, and Venkatesh Raman. SAT-based analysis of large real-world feature models is easy. In *Proceedings of the 19th International Conference on Software Product Line*, pages 91–100. ACM, 2015.

[LHCF+13] Roberto Erick Lopez-Herrejon, Francisco Chicano, Javier Ferrer, Alexander Egyed, and Enrique Alba. Multi-objective optimal test suite computation for software product line pairwise testing. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 404–407. IEEE, 2013.

[LHFRE15] Roberto E. Lopez-Herrejon, Stefan Fischer, Rudolf Ramler, and Alexander Egyed. A first systematic mapping study on combinatorial interaction testing for software product lines. In *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, number Iwct, pages 1–10. IEEE, 2015.

[LHLE15] Roberto E. Lopez-Herrejon, Lukas Linsbauer, and Alexander Egyed. A systematic mapping study of search-based software engineering for software product lines. *Information and Software Technology*, 61(0):33–51, 2015.

[Man02] Mike Mannion. Using First-Order Logic for Product Line Model Validation. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 176–187, London, UK, UK, 2002. Springer-Verlag.

[MBC09] Marcilio Mendonca, Moises Branco, and Donald Cowan. SPLOT: software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 761–762. ACM, 2009.

[MBD+16] Andreas Metzger, Andreas Bayer, Daniel Doyle, Amir Molzam Sharifloo, Klaus Pohl, and Florian Wessling. Coordinated run-time adaptation of variability-intensive systems: An application in cloud computing. In *Proceedings - 1st International Workshop on Variability and Complexity in Software Design, VACE 2016*, pages 5–11. Association for Computing Machinery, Inc, 2016.

[MCHB11] Raphael Michel, Andreas Classen, Arnaud Hubaux, and Quentin Boucher. A formal semantics for feature cardinalities in feature diagrams. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 82–89, New York, USA, 2011. ACM.

[MD16] Leticia Montalvillo and Oscar Díaz. Requirement-driven evolution in software product lines: A systematic mapping study. *J. Syst. Softw.*, 122:110–143, 2016.

[MDSD14] Raúl Mazo, Cosmin Dumitrescu, Camille Salinesi, and Daniel Diaz. Recommendation heuristics for improving product line configuration processes. In *Recommendation Systems in Software Engineering*, pages 511–537. 2014.

[MFTR+15] Juan C. Muñoz-Fernández, Gabriel Tamura, Irina Raicu, Raúl Mazo, and Camille Salinesi. REFAS: a PLE approach for simulation of self-adaptive systems requirements. In *Proceedings of the 19th International Conference on Software Product Line - SPLC '15*, pages 121–125, New York, USA, 2015. ACM Press.

[MGH+11] Raúl Mazo, Paul Grünbacher, Wolfgang Heider, Rick Rabiser, Camille Salinesi, and Daniel Diaz. Using constraint programming to verify DOPLER variability models. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '11*, pages 97–103, New York, USA, 2011. ACM Press.

[MGSH13] Dusica Marijan, Arnaud Gotlieb, Sagar Sen, and Aymeric Hervieu. Practical pairwise testing for software product lines. In *Proceedings of the 17th international software product line conference*, pages 227–235. ACM, 2013.

[MH69] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[MHG+11] Bardia Mohabbati, Marek Hatala, Dragan Gašević, Mohsen Asadi, and Marko Bošković. Development and configuration of service-oriented systems families. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 1606–1613. ACM, 2011.

[MLHS+11] Raúl Mazo, Roberto E. Lopez-Herrejon, Camille Salinesi, Daniel Diaz, and Alexander Egyed. Conformance Checking with Constraint Logic Programming: The Case of Feature Models. In *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual*, pages 456–465. IEEE, Ieee, jul 2011.

[MMFR+15] Raúl Mazo, Juan C. Muñoz-Fernández, Luisa Rincón, Camille Salinesi, and Gabriel Tamura. VariaMos: an extensible tool for engineering (dynamic) product lines. In *Proceedings of the 19th International Conference on Software Product Line - SPLC '15*, pages 374–379, New York, New York, USA, 2015. ACM, ACM Press.

[MOP+19] Daniel-Jesus Munoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don Batory. Uniform Random Sampling Product Configurations of Feature Models That Have Numerical Features. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, volume A, pages 289–301, New York, NY, USA, sep 2019. ACM.

[MP14] Andreas Metzger and Klaus Pohl. Software product line engineering and variability management: achievements and challenges. In *Proceedings of the on Future of Software Engineering - FOSE 2014*, number June, pages 70–84, New York, New York, USA, 2014. ACM Press.

[MPH+07] A. Metzger, K. Pohl, P. Heymans, P. 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 2007)*, pages 243–253, Oct 2007.

[MRM+12] Swarup Mohalik, S Ramesh, Jean-Vivien Millo, Shankara Narayanan Krishna, and Ganesh Khandu Narwane. Tracing SPLs precisely and efficiently. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*, pages 186–195, 2012.

[MSD11] Raúl Mazo, Camille Salinesi, and Daniel Diaz. Abstract Constraints: A General Framework for Solver-Independent Reasoning on Product Line Models. *INSIGHT-Journal of International Council on Systems Engineering (INCOSE)*, 14(4):22, 2011.

[MSD12a] Raúl Mazo, Camille Salinesi, and Daniel Diaz. VariaMos: a Tool for Product Line Driven Systems Engineering with a Constraint Based Approach. In *24th International Conference on Advanced Information Systems Engineering (CAiSE Forum'12)*, Gdansk, Poland, June 2012.

[MSD+12b] Raúl Mazo, Camille Salinesi, Daniel Diaz, Olfa Djebbi, and Alberto Lora-Michiels. Constraints: The Heart of Domain and Application Engineering in the Product Lines Engineering Strategy. *International Journal of Information System Modeling and Design*, 3(2):33–68, 2012.

[MSDL11] Raúl Mazo, Camille Salinesi, Daniel Diaz, and Alberto Lora-Michiels. Transforming attribute and clone-enabled feature models into constraint programs over finite domains. In *ENASE 2011 - Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering, Beijing, China, 8-11 June, 2011.*, pages 188–199, 2011.

[MTS+14] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, and Gunter Saake. An Overview on Analysis Tools for Software Product Lines. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, SPLC '14, pages 94–101, New York, NY, USA, 2014. ACM.

[MVSD] Raúl Mazo, Angela Villota, Camille Salinesi, and Daniel Diaz. Medic: Method to diagnose inconsistent product line models using constraint graphs.

[MWC09] Marcilio Mendonca, Andrzej Wąsowski, and Krzysztof CzKarnecki. SAT-based analysis of feature models is easy. In *Proceedings of the 13th International Software Product Line Conference*, pages 231–240. Carnegie Mellon University, 2009.

[MWCC08] Marcilio Mendonca, Andrzej Wąsowski, Krzysztof Czarnecki, and Donald 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.

[MZM+14] Jabier Martinez, Tewfik Ziadi, Raul Mazo, Tegawende F. Bissyande, Jacques Klein, and Yves Le Traon. Feature Relations Graphs: A Visualisation Paradigm for Feature Constraints in Software Product Lines. In *Second IEEE Working Conference on Software Visualization*, pages 50–59. IEEE, IEEE, sep 2014.

[NBE12] Alexander Nöhrer, Armin Biere, and Alexander Egyed. Managing SAT inconsistencies with HUMUS. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, pages 83–91. ACM, 2012.

[NC16] Juan Carlos Navarro and Jaime Chavarriaga. Using microsoft solver foundation to analyse feature models and configurations. In Rodriguez Y.A., editor, *2016 8th Euro American Conference on Telematics and Information Systems, EATIS 2016*. Institute of Electrical and Electronics Engineers Inc., 2016.

[NN16] Damir Nešić and Mattias Nyberg. Multi-view modeling and automated analysis of product line variability in systems engineering. In *Proceedings of the 20th International Systems and Software Product Line Conference*, SPLC '16, pages 287–296, New York, NY, USA, 2016. ACM.

[NSB+07] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, pages 529–543, 2007.

[OGRT15] Lina Ochoa, Oscar González-Rojas, and Thomas Thüm. Using decision rules for solving conflicts in extended feature models. In Di Ruscio D Volter M. Paige R., editor, *SLE 2015 - Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, pages 149–160. Association for Computing Machinery, Inc, 2015.

[Par76] D.L. Parnas. On the Design and Development of Program Families. *IEEE Transactions on Software Engineering*, SE-2(1):1–9, mar 1976.

[PBD+12] Andreas Pleuss, Goetz Botterweck, Deepak Dhungana, Andreas Polzer, and Stefan Kowalewski. Model-driven support for product line evolution on feature level. *J. Syst. Softw.*, 85(10):2261–2274, October 2012.

[PBvdL05] Klaus Pohl, Günter Böckle, and Frank van der Linden. *Software Product Line Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[PFMM08] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic Mapping Studies in Software Engineering. *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, pages 68–77, 2008.

[Plo81] Gordon D. Plotkin. A Structural Approach to Operational Semantics. Technical report, University of Aarhus, Denmark, 1981.

[PLP11] Richard Pohl, Kim Lauenroth, and Klaus Pohl. A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 313–322. IEEE Computer Society, Ieee, nov 2011.

[PMI13] PMI. *Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Project Management Institute (PMI), Newtown Square, PA, USA, 5 edition, 2013.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57. IEEE Computer Society, 1977.

[PRM+12] Carlos Parra, Daniel Romero, Sébastien Mosser, Romain Rouvoy, Laurence Duchien, and Lionel Seinturier. Using constraint-based optimization and variability to support continuous self-adaptation. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 486–491. ACM, 2012.

[psG] pure-systems GmbH. Technical white paper variant management with pure::variants.

[PSK+10] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. Automated and scalable t-wise test case generation strategies for software product lines. In *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, pages 459–468. IEEE, 2010.

[PVK15] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[QPB+14] Clément Quinton, Andreas Pleuss, Daniel Le Berre, Laurence Duchien, and Goetz Botterweck. Consistency checking for the evolution of cardinality-based feature models. In *Proceedings of the 18th International Software Product Line Conference-Volume 1*, pages 122–131. ACM, 2014.

[QRD13] Clément Quinton, Daniel Romero, and Laurence Duchien. Cardinality-based feature models with constraints. In *Proceedings of the 17th International Software Product Line Conference on - SPLC '13*, page 162, New York, New York, USA, 2013. ACM Press.

[RAC+98] C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N. Maiden, M. Jarke, P. Haumer, K. Pohl, E. Dubois, and P. Heymans. A proposal for a scenario classification framework. *Requirements Engineering*, 3(1):23–47, 1998.

[RBSW11] Iris Reinhartz-Berger, Arnon Sturm, and Yair Wand. External Variability of Software: Classification and Ontological Foundations. In Manfred Jeusfeld, Lois Delcambre, and Tok-Wang Ling, editors, *Conceptual Modeling – ER 2011*, volume 6998 LNCS, pages 275–289. Springer Berlin Heidelberg, 2011.

[RFBC10] Fabricia Roos-Frantz, David Benavides, and Antonio Ruiz Cortés. Automated Analysis of Orthogonal Variability Models using Constraint Programming. In *JISBD*, pages 269–280, 2010.

[RFBRC+12] Fabricia Roos-Frantz, David Benavides, Antonio Ruiz-Cortés, André Heuer, and Kim Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal*, 20(3-4):519–565, 2012.

[RGHB21] David Romero, José Á. Galindo, Jose-Miguel Horcas, and David Benavides. *A First Prototype of a New Repository for Feature Model Exchange and Knowledge Sharing*, page 80–85. Association for Computing Machinery, New York, NY, USA, 2021.

[RGM+15] Luisa Rincón, Gloria Giraldo, Raúl Mazo, Camille Salinesi, and Daniel Diaz. Method to Identify Corrections of Defects on Product Line Models. *Electronic Notes in Theoretical Computer Science*, 314:61–81, 2015.

[RR13] Robert Röβger and Georg Rock. A Framework and Generator for Large Parameterized Feature Models. *ISPE - International Conference on Concurrent Engineering*, 2013.

[RRIG09] Jan Recker, Michael Rosemann, Marta Indulska, and Peter Green. Business Process Modeling A Comparative Analysis. *Journal of the Association for Information Systems*, 10(4):1, 2009.

[RvBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., 2006.

[SDD+09] Camille Salinesi, Daniel Diaz, Olfa Djebbi, Raúl Mazo, and Colette Rolland. Exploiting the Versatility of Constraint Programming over Finite Domains to Integrate Product Line Models. In *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International*, pages 375–376. IEEE, Ieee, aug 2009.

[SDNB04] Marco Sinnema, Sybren Deelstra, Jos Nijhuis, and Jan Bosch. Covamof: A framework for modeling variability in software product families. In Robert L. Nord, editor, *Software Product Lines*, pages 197–213, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Seg08] Sergio Segura. Automated Analysis of Feature Models Using Atomic Sets. In *SPLC (2)*, pages 201–207, 2008.

[SFE+21] Chico Sundermann, Kevin Feichtinger, Dominik Engelhardt, Rick Rabiser, and Thomas Thüm. Yet another textual variability language? a community effort towards a unified language. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume A*, SPLC '21, page 136–147, New York, NY, USA, 2021. Association for Computing Machinery.

[SGW10] Runyu Shi, Jianmei Guo, and Yinglin Wang. A preliminary experimental study on optimal feature selection for product derivation using knapsack approximation. In *Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on*, volume 1, pages 665–669. IEEE, 2010.

[SH13] Denny Schneeweiss and Petra Hofstedt. *FdConfig: A Constraint-Based Interactive Product Configurator*, pages 239–255. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[Sha98] David C Sharp. Reducing avionics software cost through component based product line development. In *Digital Avionics Systems Conference, 1998. Proceedings., 17th DASC. The AIAA/IEEE/SAE*, volume 2, pages G32–1. IEEE, 1998.

[SHTB07] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Generic semantics of feature diagrams. *Comput. Netw.*, 51(2):456–479, February 2007.

[SK10] Tripti Saxena and Gabor Karsai. Towards a generic design space exploration framework. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1940–1947. IEEE, 2010.

[SKES18] Klaus Schmid, Christian Kröher, and Sascha El-Sharkawy. Variability modeling with the integrated variability modeling language (ivml) and easy-producer. In *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1*, SPLC '18, page 306, New York, NY, USA, 2018. Association for Computing Machinery.

[SM12] Camille Salinesi and Raúl Mazo. Defects in Product Line Models and How to Identify Them. In *Software Product Line - Advanced Topics*, chapter 5, page 50. InTech, apr 2012.

[SMD10] Camille Salinesi, Raúl Mazo, and Daniel Diaz. Criteria for the verification of feature models. In *INFORSID 2010*, number i, 2010.

[SMD+11a] Camille Salinesi, Raul Mazo, Olfa Djebbi, Daniel Diaz, and Alberto Lora-Michiels. Constraints: The core of product line engineering. In *Fifth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–10, 2011.

[SMD+11b] Camille Salinesi, Raúl Mazo, Olfa Djebbi, Raúl Mazo, Daniel Diaz, and Alberto Lora-Michiels. Constraints: the core of product line engineering. *Proceedings of the Fifth {IEEE} International Conference on Research Challenges in Information Science, {RCIS} 2011, Gosier, Guadeloupe, France, 19-21 May, 2011*, pages 1–10, 2011.

[SMD+12] Pete Sawyer, Raúl Mazo, Daniel Diaz, Camille Salinesi, and Danny Hughes. Using Constraint Programming to Manage Configurations in Self-Adaptive Systems. *Computer*, 45(10):56–63, oct 2012.

[SMDD10] Camille Salinesi, Raul Mazo, Daniel Diaz, and Olfa Djebbi. Using integer constraint solving in reuse based requirements engineering. In *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, RE '10, pages 243–251, Washington, DC, USA, 2010. IEEE Computer Society.

[SMN+10] Graeme Shanks, Daniel Moody, Jasmina Nuredini, Daniel Tobin, and Ron Weber. Representing Classes of Things and Properties in General in Conceptual Modelling. *Journal of Database Management*, 21(2):1–25, apr 2010.

[SR90] Vijay A. Saraswat and Martin Rinard. Concurrent constraint programming. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '90*, pages 232–245, New York, New York, USA, 1990. ACM Press.

[SRD16] Gustavo Sousa, Walter Rudametkin, and Laurence Duchien. Extending feature models with relative cardinalities. In *Proceedings of the 20th International Systems and Software*

Product Line Conference, SPLC '16, page 79–88, New York, NY, USA, 2016. Association for Computing Machinery.

[SRD17] Gustavo Sousa, Walter Rudametkin, and Laurence Duchien. Extending dynamic software product lines with temporal constraints. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '17, page 129–139. IEEE Press, 2017.

[SRM09] Camille Salinesi, Colette Rolland, and Raúl Mazo. VMWare: Tool support for automatic verification of structural and semantic correctness in product line models. In *International Workshop on Variability Modelling of Software-intensive Systems (VaMos)*, page 173, 2009.

[SS02] Morten Heine Sørensen and Jens Peter Secher. From type inference to configuration. In *The Essence of Computation: Complexity, Analysis, Transformation*, page 436–471, Berlin, Heidelberg, 2002. Springer-Verlag.

[STJ09] Frans Sanen, Eddy Truyen, and Wouter Joosen. Mapping problem-space to solution-space features: a feature interaction approach. In *ACM Sigplan Notices*, volume 45, pages 167–176. ACM, 2009.

[SWK+16] Thomas Schnabel, Markus Weckesser, Roland Kluge, Malte Lochau, and Andy Schürr. Cardygan: Tool support for cardinality-based feature models. In de Almeida E S Alves V. Schaefer I., editor, *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, volume 27-29-January, pages 33–40. Association for Computing Machinery, 2016.

[SYP01] Giancarlo Succi, Jason Yip, and Witold Pedrycz. Holmes: an intelligent system to support software product line development. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 829–830. IEEE Computer Society, 2001.

[SZFW05] Jing Sun, Hongyu Zhang, Yuan Fang, and Hai Wang. Formal semantics and verification for feature modeling. In *Engineering of Complex Computer Systems, 2005. ICECCS 2005. Proceedings. 10th IEEE International Conference on*, pages 303–312. IEEE, 2005.

[TBD+08] Pablo Trinidad, David Benavides, Amador Durán, Antonio Ruiz-Cortés, and Miguel Toro. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, 81(6):883–896, 2008.

[TBG13] Leopoldo Teixeira, Paulo Borba, and Rohit Gheyi. Safe composition of configuration knowledge-based software product lines. *Journal of Systems and Software*, 86(4):1038 – 1053, 2013.

[TBK09] Thomas Thüm, Don Batory, and Christian Kästner. Reasoning about edits to feature models. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 254–264. IEEE, 2009.

[TBKC07] Sahil Thaker, Don Batory, David Kitchin, and William Cook. Safe composition of product lines. In *Proceedings of the 6th international conference on Generative programming and component engineering*, pages 95–104. ACM, 2007.

[tBLLLV15a] Maurice H. ter Beek, Axel Legay, Alberto Lafuente Lluch, and Andrea Vandin. Quantitative analysis of probabilistic models of software product lines with statistical model checking. In *Proceedings 6th Workshop on Formal Methods and Analysis in SPL Engineering FMSPLE*, pages 56–70, 2015.

[tBLLLV15b] Maurice H. ter Beek, Axel Legay, Alberto Lluch Lafuente, and Andrea Vandin. Statistical analysis of probabilistic models of software product lines with quantitative constraints. In *Proceedings of the 19th International Conference on Software Product Line*, volume 20-24-July, pages 11–15. Association for Computing Machinery, 2015.

[tBLLLV16] Maurice H. ter Beek, Axel Legay, Alberto Lluch Lafuente, and Andrea Vandin. *Statistical model checking for product lines*, volume 9952 LNCS, pages 114–133. Springer Verlag, 2016.

[tBLP13] Maurice H ter Beek, Alberto Lluch Lafuente, and Marinella Petrocchi. Combining declarative and procedural views in the specification and analysis of product families. In *Proceedings of the 17th International Software Product Line Conference Co-located Workshops*, pages 10–17. ACM, 2013.

[TBRC06] Pablo Trinidad, David Benavides, and Antonio Ruiz-Cortés. Isolated Features Detection in Feature Models. In *CAiSE*, 2006.

[tBSE19] Maurice H. ter Beek, Klaus Schmid, and Holger Eichelberger. Textual variability modeling languages. In *Proceedings of the 23rd International Systems and Software Product Line Conference volume B - SPLC '19*, pages 1–7, New York, New York, USA, 2019. ACM Press.

[TC09] Pablo Trinidad and Antonio Ruiz Cortés. Abductive Reasoning and Automated Analysis of Feature Models: How are they connected?. *VaMos*, 9:145–153, 2009.

[TC16] Thammasak Thianniwet and Myra B. Cohen. *Scaling up the fitness function for reverse engineering feature models*, volume 9962 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 128–142. Springer Verlag, 2016.

[TCO00] Peter Toft, Derek Coleman, and Joni Ohta. A cooperative model for cross-divisional product development for a software product line. In *Proceedings of the First Conference on Software Product Lines : Experience and Research Directions: Experience and Research Directions*, pages 111–132, Norwell, MA, USA, 2000. Kluwer Academic Publishers.

[TKB+14] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming*, 79:70–85, jan 2014.

[TMV+16] Anna Tidstam, Johan Malmqvist, Alexey Voronov, Knut Å kesson, and Martin Fabian. Formulating constraint satisfaction problems for the inspection of configuration rules. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM).*, 30(3):313–328, 2016.

[TSS19] Thomas Thüm, Christoph Seidl, and Ina Schaefer. On language levels for feature modeling notations. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B*, SPLC '19, pages 158–161, New York, NY, USA, 2019. ACM.

[UGKB08] Engin Uzuncaova, Daniel Garcia, Sarfraz Khurshid, and Don Batory. Testing software product lines using incremental test generation. In *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, pages 249–258. IEEE, 2008.

[UKB10] Engin Uzuncaova, Sarfraz Khurshid, and Don Batory. Incremental test generation for software product lines. *Software Engineering, IEEE Transactions on*, 36(3):309–322, 2010.

[VK02] Arie Van Deursen and Paul Klint. Domain-specific language design requires feature descriptions. *CIT. Journal of computing and information technology*, 10(1):1–17, 2002.

[VMS18] Angela Villota, Raúl Mazo, and Camille Salinesi. On the Ontological Expressiveness of the High-Level Constraint Language for Product Line Specification. In *System Analysis and Modeling. Languages, Methods, and Tools for Systems Engineering*, number 1, pages 46–66, Copenhagen, 2018. Springer.

[VMS19] Angela Villota, Raúl Mazo, and Camille Salinesi. The high-level variability language: An ontological approach. In *3rd International Systems and Software Product Line Conference - Volume B (SPLC ?19)*, pages 46–66, New York, NY, USA,, 2019. ACM.

[VRH04] Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. The MIT Press, 1st edition, 2004.

[Wan10] Shige Wang. Domain-Specific Feature Modeling for High Integrity Vehicle Control System Functional Design. In *High-Assurance Systems Engineering (HASE), 2010 IEEE 12th International Symposium on*, pages 142–151. IEEE, 2010.

[WBS+10] Jules White, David Benavides, Douglas C Schmidt, Pablo Trinidad, Brian Dougherty, and Antonio Ruiz-Cortés. Automated diagnosis of feature model configurations. *Journal of Systems and Software*, 83(7):1094–1107, 2010.

[WDSB09] Jules White, Brian Dougherty, Doulas C Schmidt, and David Benavides. Automated reasoning for multi-step feature model configuration problems. In *Proceedings of the 13th International Software Product Line Conference*, pages 11–20. Carnegie Mellon University, 2009.

[WGS+14] Jules White, José A Galindo, Tripti Saxena, Brian Dougherty, David Benavides, and Douglas C Schmidt. Evolving feature model configurations in software product lines. *Journal of Systems and Software*, 87:119–136, 2014.

[WLS+16] Markus Weckesser, Malte Lochau, Thomas Schnabel, Björn Richerzhagen, and Andy Schürr. *Mind the gap! automated anomaly detection for potentially unbounded cardinality-based feature models*, volume 9633 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 158–175. Springer Verlag, 2016.

[WMMR05] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Requir. Eng.*, 11(1):102–107, December 2005.

[WN09] Lin Wang and Wee Keong Ng. Semantic modeling for DCSP-based product configuration. In *TENCON 2009-2009 IEEE Region 10 Conference*, pages 1–6. IEEE, 2009.

[WNS09] Lin Wang, Wee Keong Ng, and Bing Song. Constraint Satisfaction Approach on Product Configuration with Cost Estimation. In Been-Chian Chien, Tzung-Pei Hong, Shyi-Ming Chen, and Moonis Ali, editors, *Next-Generation Applied Intelligence*, pages 731—-740. Springer Berlin Heidelberg, 2009.

[WSB+08] Jules White, Douglas C Schmidt, David Benavides, Pablo Trinidad, and Antonio 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.

[WSC+07] Jules White, Doulas C Schmidt, Krzysztof Czarnecki, Christoph Wienands, and Gunther Lenz. Automated model-based configuration of enterprise java applications. In *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*, page 301. IEEE, 2007.

[WSWN07] Jules White, Douglas C Schmidt, Egon Wuchner, and Andrey Nechypurenko. Automating product-line variant selection for mobile devices. In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 129–140. IEEE, 2007.

[WSWN08] Jules White, Douglas C Schmidt, Egon Wuchner, and Andrey Nechypurenko. Automatically composing reusable software components for mobile devices. *Journal of the Brazilian Computer Society*, 14(1):25–44, 2008.

[WW93] Yair Wand and Ron Weber. On the ontological expressiveness of information systems analysis and design grammars. *Information Systems Journal*, 3(4):217–237, oct 1993.

[WZZ09] Xiaoguo Wang, Jin Zheng, and Qian Zeng. A design of product collaborative online configuration model. In *Cooperative Design, Visualization, and Engineering*, pages 359–366. Springer, 2009.

[WZZ15] Lin Wang, Shi-Sheng Zhong, and Yong-Jian Zhang. Process configuration based on generative constraint satisfaction problem. *Journal of Intelligent Manufacturing*, pages 1–13, 2015.

[XHSC12] Yingfei Xiong, Arnaud Hubaux, Steven She, and Krzysztof Czarnecki. Generating range fixes for software configuration. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 58–68. IEEE, 2012.

[Xte] Xtext. Xtext, language engineering for everyone.

[YZZM09] Hua Yan, Wei Zhang, Haiyan Zhao, and Hong Mei. An optimization strategy to feature models' verification by eliminating verification-irrelevant features and constraints. In *Formal Foundations of Reuse and Domain Engineering*, pages 65–75. Springer, 2009.

[ZC] Zippel and Contributors. kconfig-language.txt.

[ZKY+14] Ed Zulkoski, Chris Kleynhans, Ming-Ho Yee, Derek Rayside, and Krzysztof Czarnecki. Optimizing alloy for multi-objective software product line configuration. In *Proceedings of 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z. Springer*, pages 328–333. Springer, 2014.

[ZMZ06] Wei Zhang, Hong Mei, and Haiyan Zhao. Feature-driven requirement dependency analysis and high-level software design. *Requirements Engineering*, 11(3):205–220, 2006.

[ZYZJ08] Wei Zhang, Hua Yan, Haiyan Zhao, and Zhi Jin. A bdd-based approach to verifying clone-enabled feature models' constraints and customization. In *High Confidence Software Reuse in Large Systems*, pages 186–199. Springer, 2008.

[ZZM04] Wei Zhang, Haiyan Zhao, and Hong Mei. A propositional logic-based method for verification of feature models. In *Formal Methods and Software Engineering*, pages 115–130. Springer, 2004.

[ZZM11] Wei Zhang, Haiyan Zhao, and Hong Mei. Binary-search based verification of feature models. In *Top Productivity through Software Reuse*, pages 4–19. Springer, 2011.