IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 8 September 2022, accepted 18 December 2022, date of publication 21 December 2022,
date of current version 29 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3231449

RESEARCH ARTICLE

Exploring Functionality and Efficiency of Feature Model Product Configuration Solutions

CRISTIAN VIDAL-SILVA[1], JESENNIA CARDENAS-COBO[2], AURORA SÁNCHEZ ORTIZ[3], VANNESSA DUARTE[4], AND MIGUEL TUPAC-YUPANQUI[5]
[1]Escuela de Ingeniería en Desarrollo de Videojuegos y Realidad Virtual, Facultad de Ingeniería, Universidad de Talca, Talca 3460000, Chile
[2]Facultad de Ciencias e Ingenierías, Universidad Estatal de Milagro, Milagro 091050, Ecuador
[3]Departamento de Administración, Universidad Católica del Norte, Antofagasta, Chile
[4]Escuela de Ciencias Empresariales, Universidad Católica del Norte, Coquimbo 1780000, Chile
[5]EAP Ingeniería de Sistemas e Informática, Universidad Continental, Huancayo 12001, Peru

Corresponding author: Cristian Vidal-Silva (cvidal@utalca.cl)

ABSTRACT Variability-intensive systems are software systems in which variability management is a core activity. Examples of variability-intensive systems are the web content management system Drupal, the Linux kernel, and the Linux Debian distributions. Feature models have been considered valuable tools for modeling variability-intensive systems for more than 30 years, and their automated analysis is a thriving, motivating, and active research area. In 2010, Benavides et al. published the survey results of the first 20 years of Automated Analysis of Feature Model AAFM solutions. At that time, mainly sequential computing solutions exist. The product configuration of feature models represents a relevant operation demanding efficient automated solutions, which are now possible for assisting the feature model product configuration, such as minimal conflict detection, diagnosis, and product completion. The two main goals of this article are the following: First, to review the fundaments of product configuration of feature models. Second, to assess the functionality and computing performance of commonly used AAFM solutions for minimal conflict detection, minimal diagnosis, and the minimal completion of partial product configuration and the approaches. This article summarizes research opportunities for developing new and more efficient solutions for conflict detection, diagnosis, and product completion of large-scale configurations.

INDEX TERMS Feature model, product configuration, AAFM solutions, conflict detection, diagnosis, product completion.

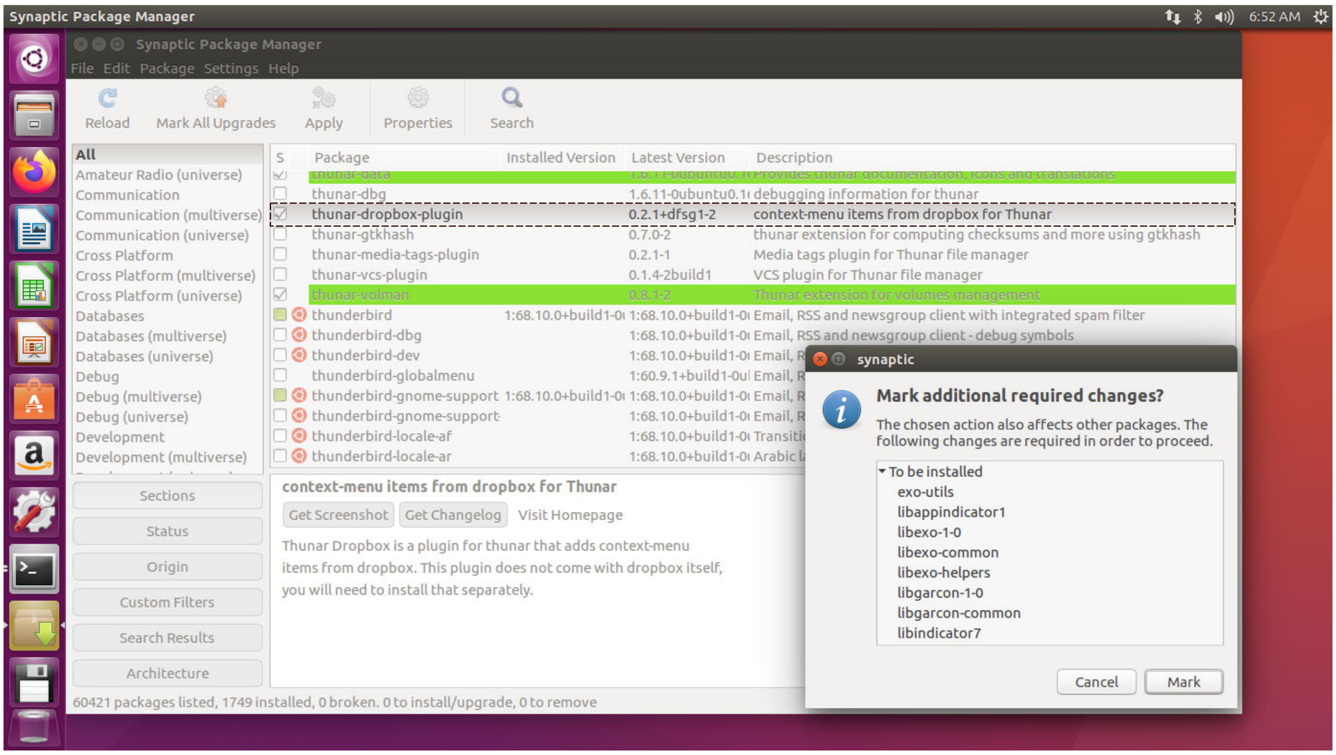
I. INTRODUCTION

Variability-Intensive Systems (VIS) are software systems in which variability management and the product configuration are core activity [1], [2]. Current examples of variability-intensive systems are the Linux kernel, the Linux Debian distributions, and the Android ecosystem.

Product configuration is the activity of designing a product according to a set of requirements and configuration rules [1]. With further details, product configuration systems need the knowledge base regarding the set of components and combination rules and the customers' requirements for selecting product components (configuration) that match their preferences [3]. A valid product configuration only depends on the selected features in a consistent knowledge base scenario; that is, each valid product results from the composition of component type instances that respect the set of defined combination rules [4]. Figure 1 shows a motivating scenario of product configuration for updating the set of installed packages in the Ubuntu Xenial operating system [5]. In this example, all the packages for installing and their dependencies represent the consistent knowledge base, and already installed and selected for installing packages correspond to the desired product requirements. Hence, after selecting a package for installation, such as a dropbox plugin, that package requires installing additional packages to respect dependency rules.

Product configuration systems require systematically managing all features and composition rules to analyze the feature

The associate editor coordinating the review of this manuscript and approving it for publication was P. K. Gupta.



S	Package	Installed Version	Latest Version	Description
✓	thunar-data		1.6.11-0ubuntu0.1	Provides thunar documentation, icons and translations
	thunar-dbg		1.6.11-0ubuntu0.1	debugging information for thunar
✓	thunar-dropbox-plugin		0.2.1+dfsg1-2	context-menu items from dropbox for Thunar
	thunar-gtkhash		0.7.0-2	thunar extension for computing checksums and more using gtkhash
	thunar-media-tags-plugin		0.2.1-1	Media tags plugin for Thunar file manager
	thunar-vcs-plugin		0.1.4-2build1	VCS plugin for Thunar file manager
✓	thunar-volman		0.8.1-2	Thunar extension for volumes management
🟩	thunderbird	1:68.10.0+build1-0	1:68.10.0+build1-0	Email, RSS and newsgroup client with integrated spam filter
	thunderbird-dbg		1:68.10.0+build1-0	Email, RSS and newsgroup client - debug symbols
	thunderbird-dev		1:68.10.0+build1-0	Email, R
	thunderbird-globalmenu		1:60.9.1+build1-0ul	Email, R
🟩	thunderbird-gnome-support	1:68.10.0+build1-0	1:68.10.0+build1-0	Email, R
	thunderbird-gnome-support		1:68.10.0+build1-0	Email, R
	thunderbird-locale-af		1:68.10.0+build1-0	Transiti
	thunderbird-locale-ar		1:68.10.0+build1-0	Arabic l

context-menu items from dropbox for Thunar

Thunar Dropbox is a plugin for thunar that adds context-menu items from dropbox. This plugin does not come with dropbox itself, you will need to install that separately.

60421 packages listed, 1749 installed, 0 broken. 0 to install/upgrade, 0 to remove

Mark additional required changes?

The chosen action also affects other packages. The following changes are required in order to proceed.

- To be installed
  - exo-utils
  - libappindicator1
  - libexo-1-0
  - libexo-common
  - libexo-helpers
  - libgarcon-1-0
  - libgarcon-common
  - libindicator7

FIGURE 1. Example of the packages configuration in the Ubuntu Xenial OS [1].

selection for desired products. Variability Models (VMs) permit describing the different relationships and configuration options for the variability management of software systems in software engineering. Different VMs exist, such as Feature Models (FMs) and Orthogonal Variability Models (OVMs). FMs permit representing functional commonalities and variabilities of software systems [6], whereas OVMs permit describing the variant parts of the base model of systems [7]. Kang et al. [8] introduced FMs as part of the FODA (Feature-Oriented Domain Analysis) method, and they became the most used VM in the SPL community afterwards.

An FM defines a set of features and their relationships for defining valid feature combinations or products, that is, sets of features that respect the FM's defined relationships. Such as Apel et al. [9] remark, FM permits representing all the products of an SPL. An FM organizes in a tree-like structure that starts at the root feature that identifies the SPL and from which tree branches of features emerge. We can then define a product in terms of the set of features that compose it, and each feature describes an increment of functionality in the products containing that feature [10]. An FM supports binary and set relationships between parent and child features and cross-tree relationships to symbolize dependencies between features. Figure 2 shows an FM for describing an operating system SPL and a valid configuration of it (the grey-coloured features). Figure 3 shows the same FM along with a non-valid configuration of it.

A. PROBLEM STATEMENT, GOAL AND CONTRIBUTIONS

Product configuration permits assisting the mass customization production [4]. Variability-Intensive Systems (VIS) follow mass customization in software engineering by addressing variability in the software development process phases. Because VIS users expect that software products can adapt to their needs, the management of the users' requirements variability in VIS represents an important activity for those systems [11]. Research works concerning managing the variability of VIS already exist in the literature, such as in the Linux operating system [12], [13], the Debian-based distributions of Linux [14], with the Android mobile system [15], and with the content management framework Drupal [16]. Those works use variability models for representing and analyzing VIS.

Software Product Line (SPL) is a case of VIS that systematically manages commonalities and variabilities for the configuration of software products [9]. SPL defines domain engineering as analyzing and developing reusable common and variable functionalities (features) in the products' domain and applying engineering to produce customized products regarding user feature selection. Defining valid configuration in SPL is a complex task for the growing complexity of the configuration knowledge base [15]. When the users' feature selection conflicts with consistent configuration knowledge bases, identifying those issues to solve them is necessary. Such as Benavides et al. [6] remark, the manual analysis of variability models, such as FMs, are error-prone

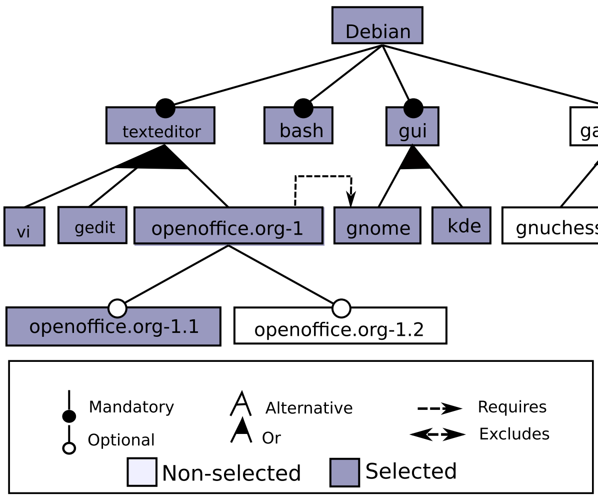


FIGURE 2. Feature model with a valid configuration example [1].

and time-consuming tasks mainly for their increasing size. For example, variability and configuration models for Debian-based distributions describe around twenty-eight thousand variability points [14], and manually analyzing those models without mistakes is impractical. Mechanisms for the Automated Analysis of Feature Models (AAFM) [6] are a solution to face those issues.

A set of features in an FM is called a configuration, and each software variant in an FM identifies a valid configuration or product [17]. FMs permit organizing the configuration space to facilitate the construction of software variants by describing configuration options using interdependent features or functionalities. Hence, AAFM operations for assisting the obtention of conflict-free FMs and configurations are high-value tasks. Nonetheless, existing Automated Analysis of Feature Model operations usually follow a sequential computing approach and cannot scale to work on large-scale and high-variability models. Various algorithms and solutions applicable for the AAFM exist in the literature, such as QuickXPlain [18], FastDiag [19] to detect a minimal conflict and minimal-preferred diagnosis in a set of constraints in conflict, respectively, and solutions for the completion of partial products. Concerning the completion of products, defining and accomplishing all users' requirements for configuring large-scale systems is a tedious and complex task, and non-efficient solutions exist to assist in completing partial configurations. Those solutions cannot use additional computing resources for their sequential computing nature, such as multiple cores or network technologies for parallel and distributed computing. Next, we describe those operations in more detail.

- Minimal conflict set. For a consistent FM that we can define as a set of constraints, a non-consistent configuration violates those constraints. The features model of Figure 2 exemplifies a consistent configuration; that is, this configuration does not violate the FM constraints. For that FM and configuration, if feature *gnuchess* were also selected, the resulting configuration would be non-consistent because a conflict exists between features *gnuchess* and *glchess*. In this example, {*gnuchess*, *glchess*} is a Minimal Conflict Set (MCS) because we cannot find a subset of it that results being a conflict set. Such as [19] remark, we can solve an MCS by merely deleting one of its constraints. After finding and solving all the MCS instances, the configuration is valid (conflicts-free).
  The functioning of QuickXPlain uses the consistency check over constraint sets, a costly action, as a primary step to achieve its main purpose: to identify a preferred MCS. The QuickXPlain algorithm efficiently finds preferred MCS regarding the order of the constraints definition. The application of QuickXPlain for the conflict analysis of large-scale FMs such as the Android mobile operating system [15], the Linux kernel [12], and the Linux distributions like Debian [14] are examples of computationally expensive tasks. That occurs mainly for the sequential nature of QuickXPlain and the high demand for computing resources such as the execution time and memory space to work with large-scale models.
- Minimal diagnosis. Given the constraints of a consistent FM and a non-consistent configuration that violates the FM constraints, a diagnosis is the set of constraints that permit a consistent configuration after removing those constraints. For the consistent configuration of Figure 3 after selecting the feature *gnuchess*, we know that {*gnuchess*, *glchess*} is a Minimal Conflict Set (MCS). Then, either deleting *gnuchess* or *glchess*, we can obtain a valid configuration (conflict-free); that is, *gnuchess* and *glchess* are examples of diagnosis.
  The functioning of FastDiag uses the consistency check over constraint sets, a costly action, as a primary step to achieve its main purpose: to identify a preferred and minimal diagnosis. The FastDiag algorithm efficiently

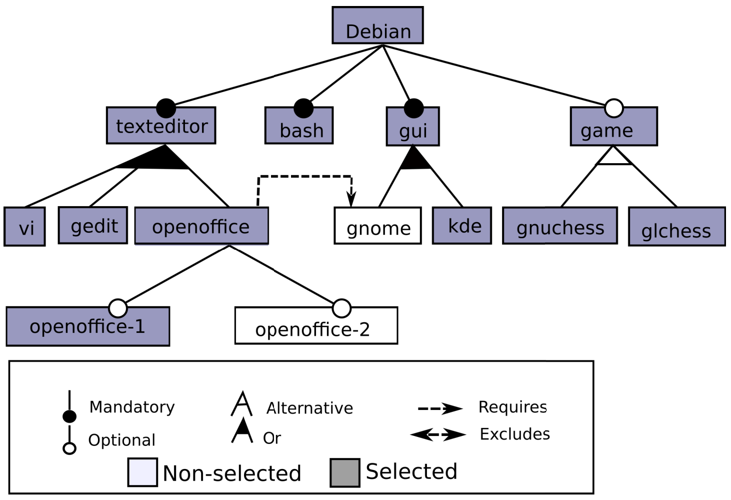


FIGURE 3. Feature model with a non-valid configuration example [1].

finds a preferred-minimal diagnosis regarding the order of the constraints definition. The application of this algorithm for the diagnosis analysis of large-scale FMs such as the Android mobile operating system [15], the Linux kernel [12], and distributions of Linux like Debian [14] result in computationally expensive tasks. That occurs mainly for the sequential nature of FastDiag and the high demand for computing resources such as the execution time and memory space to work with large-scale models.

- Minimal completion of products. Achieving valid configurations in large-scale FMs is a time-demanding and complex task. Currently, reasoning tools usually part of the AAFM process permit solving that issue.

Developing FM product configurations without conflicts requires identifying each conflict and the necessary steps to solve or diagnose them. Hence, conflict detection and diagnosis are essential operations for getting conflict-free models. Completing a product configuration of FM by hand also represents an error-prone and time-consuming task. Solutions for those tasks to work efficiently on large-scale models represent high-value tasks nowadays. Because AAFM solutions for the conflict detection, diagnosis, and completion of products already exist, the main goal of this article is to review the functionality and computing cost of existing AAFM solutions for assisting the product configuration on large-scale models. Benavides et al. [6] in 2010 presented a Systematic Literature Review (SLR) about FM and AAFM operations. Different AAFM proposals exist with the use of various formal approaches such as CSP, SAT, and BDD solvers [7]. The main contributions of this article are:

- To analyze the functionality and computing performance of commonly used AAFM solutions for minimal conflict detection, minimal diagnosis, and the minimal completion of partial product configuration.
- To describe a computing approach that can improve the performance of AAFM solutions.

This article summarizes PhD research studies of the first author to highlights research opportunities for developing new and more efficient solutions for the conflict detection, diagnosis and product completion of large-scale configurations. The rest of this paper is organized as follows. Section II describes and exemplifies the use of FMs. Section III describe the Automated Analysis of Feature Model (AAFM) and the product configuration processes. That section also details the conflict detection, diagnosis and product completion operations and existing solutions for those purposes. Section IV presents practical results of the analyzed solutions for the product configuration and partial completion for a test set. Section VI details a few practical issues of our research. The paper concludes by summarizing the benefits of our academic experience and detailing the motivation for continuing with it in the current and future years.

II. BACKGROUND

A feature model is an information model that permits representing the variant flexibility and maintainability for systems' variability, and configuration [20]. A feature is an abstraction of a prominent or distinctive user-visible aspect, requirement, quality, or functional characteristic of a family of software systems [6], [21], [22], that is, each feature constitutes a user-visible configuration option of the problem domain [23]. An FM is a tree-like structure commonly used to represent common and variable functionalities (features) and their relationships to the configuration of products in a Software Product Line (SPL) [8]. Kang et al. [8] introduced FMs in the FODA (Feature-Oriented Domain Analysis) method, and they are the "de facto" standard for describing common and variable features in system families [24], [25] regardless of their size because FMs facilitate the software reuse [26].

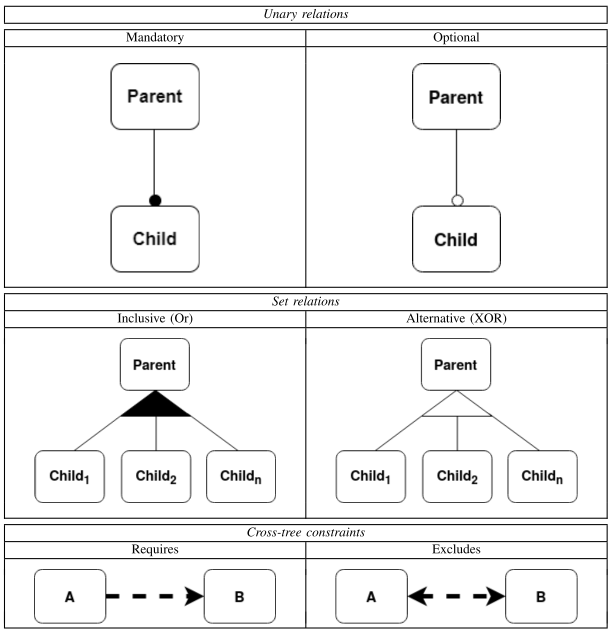
An FM starts with the root feature. Each successively deeper level in the FM corresponds to a more fine-grained configuration option for product-line variants. Features are nodes of that tree, and their relationships are the edges (relationships and constraints) between features [27]. The relationships among features are of two types: structural relationships between a parent and its child features and cross-tree or cross-hierarchy constraints [4], [6]. FMs represent an effective communication medium between customers and developers of SPLs [28]. Such as the work of Benavides et al. describe [6], different FM dialects exist nowadays such as basic FMs models [4], [10], cardinality based FMs [29], [30] and extended FMs using feature attributes [31], [32].

1) BASIC FEATURE MODELS

A basic FM support two types of relationships between features: structural relationships between parents and their child features and cross-tree constraints [4], [6]. Thus, each non-root feature has a parent feature and is either part of a group or not [33]. The next lines describe each type of FM relationship.

- Structural relationships between parents and their child features:
  - Mandatory: A mandatory relationship states that a parent feature requires its child. The top-left figure of Table 1 shows the graphic representation of a mandatory relationship between a parent feature and a child feature.
  - Optional: An optional relationship states that a child feature may be or not be present (its parent feature does not require it). The top-right figure of Table 1 illustrates an optional relationship between a parent feature and a child feature.
  - Set: A defined number of features of a set of children's features (sub-features) are selectable for products when their parent is selected. A cardinality relation [x, y] gives this number of features for x <= y and y <= number of child features in the set. Two cases are XOR (alternative) and Or (inclusive) sets.
    - ∗ Inclusive Or: At least one child's features must be present. The cardinality relation is [1, n] in this case (n corresponds to the number of child features). The middle-left figure of Table 1 illustrates an inclusive relationship between a parent feature and a set of children's features.
      The middle-right figure of Table 1 illustrates an alternative relationship between a parent feature and a set of children's features.
    - ∗ Alternative XOR: Only one child feature must be present. The associated cardinality relation is [1, 1] in this case.
- Cross-Tree Constraints.
  - Requires: For two features A and B, if A requires B, then A's presence implies the presence of B in a product. The bottom-left figure of Table 1 illustrates a required cross-tree constraint relationship between a source feature A and a target feature B.
  - Excludes: For two features, A and B, if A excludes B, then A and B cannot be present in the same product. The bottom-right figure of Table 1 illustrates an excludes cross-tree constraint relationship between features A and B.

TABLE 1. Feature model relations.



Such as Benavides et al. [6] indicate that more complex cross-tree relationships exist in the literature to define constraints in the form of generic propositional formulas such as "A and not B implies C".

Figure 1 illustrates a valid configuration for the FM of the Debian operating system: nodes represent the features of the model (i.e., selectable packages to install) and edges are the constraints between features (e.g., packages that require the installation of other packages). In this example, we can observe that packages *texteditor*, *bash* and *gui* are mandatory (i.e., they must be always included in any Debian configuration) whereas the package *games* is optional. We can also observed that *textditor* requires at least one of the packages *vi*, *gedit* or *openoffice*. Likewise, feature *gui* requires at least one of *gnome* or *kde*. In the case of the package *games*, we observe that it requires either *gnuchess* or *glchess*, but only one, non both. Similarly, the package *openoffice.org-1* requires either version *openoffice.org-1.1* or *openoffice.org-1.2*. Finally, we observe that *openoffice.org-1* strictly requires the installation of *gnome*. Hence, the selection of features {*Debian*, *texteditor*, *vi*, *gedit*, *openoffice*, *openoffice-1*, *bash*, *gui*, *gnome*, *kde*, *game*, *glchess*} exemplifies a valid product. Figure 2 illustrates a non-valid configuration for the FM of the Debian operating system: the selection of features

{*Debian*, *texteditor*, *vi*, *gedit*, *openoffice*, *openoffice-1*, *bash*, *gui*, *kde*, *game*, *gnuchess*, *glchess*} exemplifies a non-valid configuration that does not respect the requires cross-tree constraint between the option *openoffice.org-1* and *gnome* (only the first option is selected), and *gnuchess* and *glchess* are selected for the option *game* when only one feature must be selected (*game* represents an alternative set of features).

The application and analysis of FMs is a common approach to performing analysis tasks. Benavides et al. [6] mention that the manual analysis of FMs is a time-demanding and error-prone activity, and the Automated Analysis of Feature Models (AAFM) process permits solving those issues. The AAFM process starts by translating the FM and additional information, such as global restrictions, into a logical set of constraints. Afterwards, queries can proceed with the translated model using an off-the-shelf solver and other tools such as programming solutions, thus obtaining analysis results [34]. Figure 4 illustrates the AAFM process.

Such as Galindo et al. [34] summarize, six different variability facets exist where the AAFM is currently applied: *i*) product configuration and derivation; *ii*) testing and evolution; *iii*) reverse engineering; *iv*) multi-model variability-analysis; *v*) variability modelling, and; *vi*) variability-intensive systems. The first AAFM application results in the most traditional usage of automated analysis mechanisms. This thesis aims to contribute to it.

Developing FM and product configurations without errors or conflicts requires identifying each conflict and the necessary steps to solve or diagnose them. Hence, conflict detection and diagnosis are operations needed for getting conflicts-free models. Completing a product configuration of FM by hand also represents an error-prone and time-consuming task. Solutions for those tasks to work efficiently on large-scale models represent high-value tasks nowadays. AAFM solutions for the conflict detection, diagnosis, and completion of products already exist.

III. AUTOMATED ANALYSIS OF VARIABILITY-INTENSIVE SYSTEMS

The development process of a Variability Intensive System (VIS) considers identifying and representing the system's components and relationships among those components as two core activities. The application and analysis of FMs is a common approach to performing those analysis tasks. Benavides et al. [6] mention that the manual analysis of FMs is a time-demanding and error-prone activity, and the Automated Analysis of Feature Models (AAFM) process permits solving those issues. The AAFM process starts by translating the FM and additional information, such as global restrictions, into a logical set of constraints. Afterwards, queries can proceed with the translated model using an off-the-shelf solver and other tools such as programming solutions, thus obtaining analysis results [34].

Such as Galindo et al. [34] summarize, six different variability facets exist where the AAFM is currently applied: *i*) product configuration and derivation; *ii*) testing and evolution; *iii*) reverse engineering; *iv*) multi-model variability-analysis; *v*) variability modelling, and; *vi*) variability-intensive systems. The first AAFM application results in the most traditional usage of automated analysis mechanisms. This thesis aims to contribute to it.

Developing FM and product configurations without errors or conflicts requires identifying each conflict and the necessary steps to solve or diagnose them. Hence, conflict detection and diagnosis are operations needed for getting conflicts-free models. Completing a product configuration of FM by hand also represents an error-prone and time-consuming task. Solutions for those tasks to work efficiently on large-scale models represent high-value tasks nowadays. AAFM solutions for the conflict detection, diagnosis, and completion of products already exist. The next sections describe an existing algorithm for detecting Minimal Conflict Sets (MCS), a current algorithm for detecting Minimal Diagnosis (MD), and traditional approaches to complete product configurations.

2) PRODUCT CONFIGURATION SOLUTIONS

a: MINIMAL CONFLICT SETS (MCS) DETECTION

An MCS of a system represents a minimal set of constraints in conflict. For definition 1 [4], it is necessary to identify the set of constraints B that represents a consistent background knowledge and the set of constraints C that is the suspected subject of a conflict search.

Definition 1: A set $AC = B \cup C = \{c_1, c_2, \ldots, c_n\}$ *represents the set of all constraints in the knowledge base; that is,* AC *is the union of the consistent knowledge base* B *and the suspicious set of constraints subject of conflict search* C*. Then, a conflict* $CS = \{c_a, c_b, \ldots, c_z\}$ *is a non-empty and non-consistent subset of* C*.* CS *is minimal if* $\neg\exists\ CS'$ *such that* $CS' \subset CS$ CS *is preferred if the order of its constraints follow a defined ranking of preferences.*

For the FM of Figure 3, concerning definition 1, the consistent base knowledge B is a formal definition of the FM, that is, a logic representation of the set of features and their relationships. We can detect conflict in the configuration of products for that model. For the product configuration $C =$ {*Debian*, *texteditor*, *bash*, *gui*, *game*, *vi*, *gedit*, *openoffice.org-1*, *gnome*, *kde*, *glchess*, *openoffice.org-1*}, the resulting minimal conflict set is {} because C represents a consistent configuration. For the product configuration $C =$ {*Debian*, *texteditor*, *bash*, *gui*, *game*, *vi*, *gedit*, *openoffice.org-1*, *kde*, *gnuchess*, *glchess*, *openoffice.org-1*}, the resulting preferred minimal conflict set is {*openoffice.org-1*, ¬*gnome*}}. The next lines describe the QuickXPlain algorithm for efficiently detecting preferred MCS.

QuickXPlain [18] is an efficient approach to determining a minimal conflict set. QuickXPlain receives C as the set of suspicious constraints with conflict and B as consistent constraints of the background knowledge. Then, a conflict does not exist if $B \cup C$ is consistent or C is empty. On the other hand, QuickXPlain proceeds by returning the results of the function QX. QX receives the parameters C

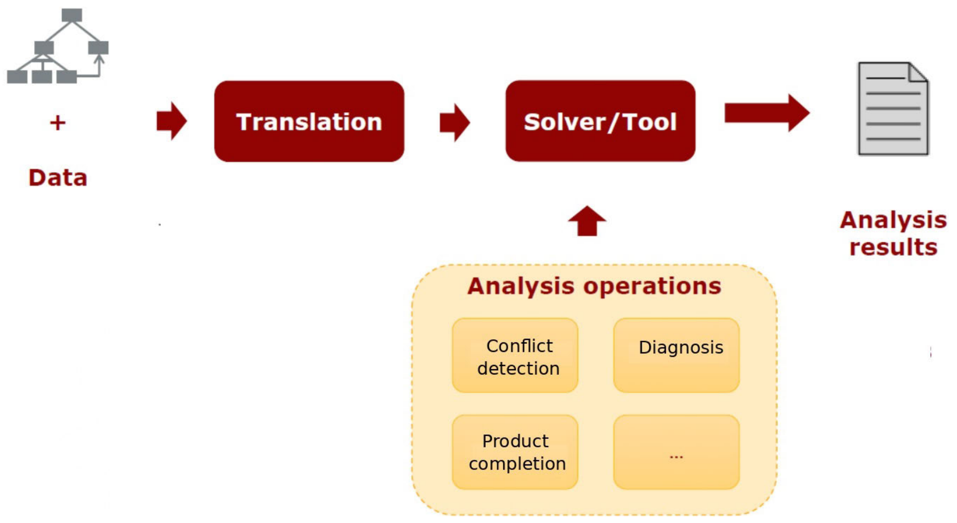


FIGURE 4. Automated Analysis of Feature Models (AAFM) process [1].

Algorithm 1 QuickXPlain$(C, B) : CS$

```
1:  if Consistent($B \cup C$) then
2:      return('no conflict')
3:  else if $C = \emptyset$ then
4:      return($\emptyset$)
5:  else
6:      return(QX($C, B, \emptyset$))
7:  end if
```

Algorithm 2 QX$(C = \{c_1..c_m\}, B, B\delta) : CS$

```
1:  if $B\delta \neq \emptyset$ and InConsistent($B$) then
2:      return($\emptyset$)
3:  end if
4:  if $C = \{c_\alpha\}$ then
5:      return($\{c_\alpha\}$)
6:  end if
7:  $k = \lfloor \frac{m}{2} \rfloor$
8:  $C_a \leftarrow c_1 \ldots c_k$;$C_b \leftarrow c_{k+1} \ldots c_m$;
9:  $\Delta_2 \leftarrow$ QX($C_a, B \cup C_b, C_b$);
10: $\Delta_1 \leftarrow$ QX($C_b, B \cup \Delta_2, \Delta_2$);
11: return($\Delta_1 \cup \Delta_2$)
```

(initially the complete set of constraints with conflict), B (initially the knowledge base), and $B\delta$ (initially empty) that represents the last items added to B. Function QX follows a divide-and-conquer approach for conflict detection. Hence, $B\delta$ corresponds to the set of constraints added for reviewing the consistency of the knowledge base, and C is the set of constraints to continue analyzing if the current B is consistent. Algorithms 1 and 2 show the pseudo-code of the functions of QuickXPlain.

QuickXPlain permits determining one MCS per computation. Felfernig et al. [4] indicate that we need to update adequately or delete one of the constraints of an MCS for solving it, and, if the model is non-consistent yet, to apply QuickXPlain and repeat the process. When the resulting model is consistent, the updated constraints represent a diagnosis or solution for the model. Table 2 shows the tracking steps of a QuickXPlain application for the FM configuration example of Figure 3. Column B represents the consistent constraints of the base knowledge for the FM definition, and column C is the set of selected features. The final result represents a minimal conflict that we can solve by adequately updating one of its constraints. In this case, a solution is to update the state of $\neg$*gnome*.

3) MINIMAL DIAGNOSIS DETECTION

Identifying and solving conflicts one by one is necessary to obtain a conflict-free model: we need to identify a conflict first, adapt (update or eliminate) constraints of that conflict for its solution, and repeat this process until no more conflict exists, that is, until reaching a consistent model. The set of all the adapted constraints for getting a conflict-free model represents a diagnosis. Definition 2 formally defines the term diagnosis [4].

Definition 2: A set $AC = \{c_1, c_2, \ldots, c_n\}$ *represents the set of all constraints in the problem for diagnosis; that is, AC is the union of the consistent base knowledge B and the set of constraints subject of conflict search C:* $AC = B \cup C$. *Then, a diagnosis is a set of constraints* $\Delta \subseteq C$ *such that* $(B \cup C - \Delta)$ *results in a consistent or conflict-free set.* Δ *is minimal if* $\neg\exists\ \Delta'$ *such that* $\Delta' \subset \Delta$. *A minimal diagnosis is of minimal cardinality if there does not exist a minimal diagnosis* Δ' *such as* $|\Delta'| < |\Delta|$.

A minimal diagnosis for the FM configuration of Figure 3 has to consider solutions for each conflict. Hence, this example contains two diagnosis options. To get a conflict-free model, the user has to solve each diagnosis. Cases with multiple diagnosis instances exist, and determining

TABLE 2. QuickXPlain execution tracking on configuration example of Figure 3.

Step	C	B	$B\delta$	C_a	C_b	Return
1	{Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	B	$\emptyset$	{Debian, texteditor, bash, gui, game, vi, gedit}	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice-1, ¬ gnome}
2	{Debian, texteditor bash, gui, game, vi, gedit}	$B \cup$ {kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome},	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}			$\emptyset$
3	{openoffice, kde, gnuchess glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	B	$\emptyset$	{openoffice kde, gnuchess}	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice-1, ¬ gnome}
4	{openoffice, kde, gnuchess}	$B \cup$ {glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}			$\emptyset$
5	{glchess, openoffice-1, ¬openoffice-2, ¬gnome}	B	$\emptyset$	{glchess, openoffice-1}	{¬openoffice-2, ¬gnome}	{openoffice-1, ¬gnome}
6	{glchess, openoffice-1}	$B \cup$ {¬openoffice-2, ¬gnome}	{¬openoffice-2, ¬gnome}	{glchess}	{openoffice-1}	{openoffice-1}
7	{glchess}	$B \cup$ {¬openoffice-2, ¬gnome, openoffice-1}	{openoffice-1}			$\emptyset$
8	{openoffice-1}	$B \cup$ {¬openoffice-2, ¬gnome}	$\emptyset$			{openoffice-1}
9	{¬openoffice-2, ¬ gnome}	$B \cup$ {¬gnome}	{openoffice-1}	{¬openoffice-2}	{gnome}	{openoffice-1}
10	{¬openoffice-2}	$B \cup$ {openoffice-1} gnome}	{¬gnome}			$\emptyset$
11	{¬gnome}	$B \cup$ {openoffice-1}	$\emptyset$			{¬gnome}

Algorithm 3 FastDiag(C, AC) : *diagnosis* Δ

```
1: if $C = \emptyset$ or InConsistent$(AC - C)$ then
2:     return$(\emptyset)$
3: else
4:     return(FD$(\emptyset, C, AC)$)
5: end if
```

all the diagnoses can be computationally expensive. Model constraints can be in relevant order for obtaining a preferred diagnosis, then obtaining all the diagnoses to look for the preferred one is a time-demanding and lost time activity since solving one diagnosis is enough for a conflict-free model. The next lines describe the FastDiag algorithm to determine a minimal preferred diagnosis.

FastDiag algorithm permits determining a preferred or leading diagnosis concerning a previously defined relevance order of constraints in the knowledge base. FastDiag follows the algorithmic structure and reasoning of QuickXPlain for a different purpose: diagnosis detection without the calculation of MCS instances. Hence, FastDiag is based on conflict-independent search

Algorithm 4 FD$(D, C = \{c_1..c_q\}, AC)$: *diagnosis* Δ

```
1: if $D \neq \emptyset$ and Consistent$(AC)$ then
2:     return$(\emptyset)$
3: end if
4: if $|C| = 1$ then
5:     return$(C)$
6: end if
7: $k = \lfloor \frac{q}{2} \rfloor$
8: $C_a \leftarrow c_1 \ldots c_k$; $C_b \leftarrow c_{k+1} \ldots c_q$;
9: $\Delta_1 \leftarrow$ FD$(C_b, C_a, AC - C_b)$;
10: $\Delta_2 \leftarrow$ FD$(\Delta_1, C_b, AC - \Delta_1)$;
11: return$(\Delta_1 \cup \Delta_2)$
```

strategies [35]. Algorithms 3 and 4 give the pseudo-code of FastDiag functions.

Assuming that conflicts to diagnosis exist, If the conflict set C is non-empty, and AC without C is consistent, algorithm FastDiag calls and waits for the recursive results algorithm FD. FD first reviews the consistency of AC as a source of diagnosis. Because always AC contains C and does not contain D, S is the constraints set with conflicts, and D is

TABLE 3. Tracking of the FastDiag execution of configuration example of Figure 5 (part 1).

Step	D	C	AC	C_a	C_b	Return
1	∅	{Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{Debian, texteditor, bash, gui, game, vi, gedit}	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, ¬ gnome}
2	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{Debian, texteditor, bash, gui, game, vi, gedit}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit}			∅
3	∅	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice, kde, gnuchess}	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, ¬ gnome}
4	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice, kde, gnuchess}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess}			{glchess, ¬ gnome}
5	∅	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, openoffice-1}	{¬ openoffice-2, ¬ gnome}	{glchess, ¬ gnome}
6	{¬ openoffice-2, ¬ gnome}	{glchess, openoffice-1}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1}	{glchess}	{openoffice-1}	{glchess, ¬ gnome}
7	{openoffice-1}	{glchess}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess}			{glchess}
8	{glchess}	{openoffice-1}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1}			∅

empty; when D is not empty, and AC is consistent, D is the source of conflict. When that base case is not accomplished, either because D is empty (such as at the beginning) or AC is consistent (this is only possible after removing elements from $AC - D$ represents the last removed elements from AC), then AC is still in conflict, and C is a source of conflict. Then, FD reviews the size of C since if it were minimal (size 1), then C is the diagnosis. If C is not of minimal size, FD proceeds to partition C in the sets C_1 and C_2, of which the last one corresponds to the most preferred partition. Afterwards, FD calls FD over C_2, C_1 and $AC - C_2$ to review if C_2 is the diagnosis source, and if not so, to continue reviewing C_1 with that goal. Tables 3 and 4 show the tracking execution of FastDiag on the FM product configuration of Figure 3. The diagnosis corresponds to the solution of each MCS, in this case, {glchess, ¬gnome}. In column AC, the symbol Γ represents the consistent knowledge base of the FM.

A. PRODUCT COMPLETION

Completing partial configurations consists of finding the non-selected components necessary to update that permit evolving the partial setting into a complete product configuration. In FM configurations, each feature is decided to be either present or absent in the resulting products, whereas in partial configurations, some features are undecided. The completion of partial configurations is a non-trivial and computationally expensive task mainly for the FM constraints [36], a process that is usually computationally more expensive in large-scale FMs. Product

TABLE 4. Tracking of the FastDiag execution of configuration example of Figure 3 (part 2).

Step	D	C	AC	C_a	C_b	Return
8	{glchess}	{openoffice-1}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1}			$\emptyset$
9	{glchess}	{¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{¬ openoffice-2}	{¬ gnome}	{¬ gnome}
10	{¬ gnome}	{¬ openoffice-2}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1, ¬ openoffice-2}			$\emptyset$
11	$\emptyset$	{¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1, ¬ openoffice-2, ¬gnome}			{¬ gnome}

configurations can result in misconfigurations (i.e., non-valid configurations) which can impact the system availability [37]. Unavailability of the Facebook platform [38], service-level problems of Google [39], and invalid operation of Hadoop clusters [40] are known misconfiguration examples.

A usual and efficient solution for the completion of partial products is the application of reasoning tools such as CSP and SAT solvers for obtaining a set of necessary features for the completion of the partial configuration. Those solutions can be minimal, but they do not always represent the preferred configuration [41].

Figure 5 illustrates a conflict-free partial product and features for a FM minimal completion. The partial configuration presents the selection of four features, {*Debian*, *texteditor*, *bash*, *gui*} (features in background color grey). Given the rest of the model's features (background colour white and background colour green), features in background colour green represent a preferred minimal completion; that is, the set of features necessary for obtaining a preferred and minimal completion of the partial configuration. That completion per default takes the typographic order of each non-selected feature as the order of preference.

In summary, QuickXPlain, FastDiag, and solvers for completing the partial product are efficient algorithm and tool solutions for identifying MCS, minimal diagnosis and the completion of partial products. Even though they are efficient sequential-computing solutions, such as Vidal et al. [2] highlight, they are not adequate to work with large-scale FMs. The next section reviews the computing performance of those solutions.

IV. AAFM PRODUCT CONFIGURATION SOLUTIONS IN PRACTICE

Next, we summarize experiments to evaluate the computing performance of QuickXPlain, FastDiag and the product completion using a CSP tool.

QX *Complexity:* Assuming a splitting $k = \lfloor \frac{m}{2} \rfloor$ of $C = \{c_1..c_m\}$, the worst-case time complexity of QuickXPlain in terms of the number of consistency checks needed for calculating one minimal conflict is $2k \times log_2(\frac{m}{k}) + 2k$ where k is the minimal conflict set size and m represents the underlying number of constraints [18]. We should optimize the computing performance of consistency checks becuase they are the most time-consuming part of conflict detection.

FD Complexity: Assuming a splitting $d = \lfloor \frac{n}{2} \rfloor$ of $S = \{s_1..s_n\}$, the worst-case time complexity of FD in terms of the number of consistency checks needed for calculating one minimal diagnosis is $2d \times log_2(\frac{n}{d}) + 2d$ where d is the minimal diagnosis set size and n represents the underlying number of constraints [35]. The runtime performance of the underlying algorithms must be optimized because consistency checks are the most time-consuming part of diagnosis detection.

Execution Environment: For QuickXPlain and FastDiag, all experiments reported were conducted using an AMD EPYC 7571 machine equipped with a CPU with eight cores and 2.60GHz. Each core maintained up to 2 threads, which means that 16 cores could be simulated using hyper-threading. It had 64 GB of RAM.

Characteristics of the Knowledge Bases: For evaluation purposes of QuickXPlain and FastDiag, we generated configuration knowledge bases (feature models) from the publicly available Betty tool suite [42], which allows for sys-

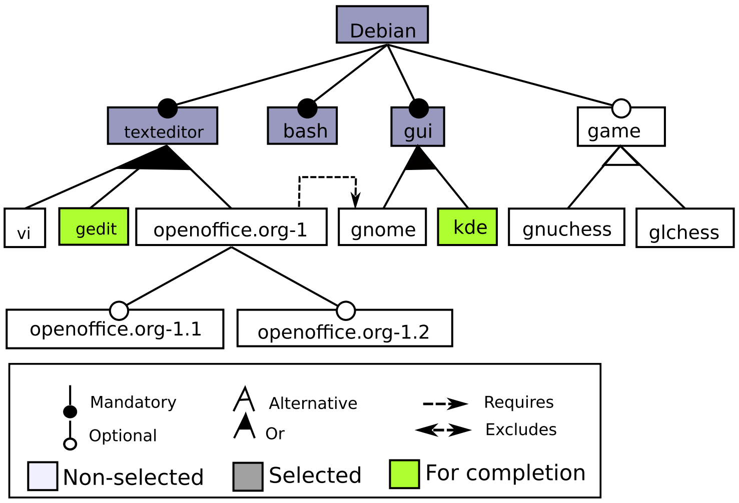


FIGURE 5. Example of a partial product and features for completion in a Debian derivative example [1].

tematic testing of different consistency checking and conflict detection approaches for knowledge bases. The knowledge base instances (represented as background knowledge B in QuickXPlain and AC in FastDiag) that were selected for the purpose of our evaluation had around 1.000 binary variables (derived from the 1.000 features used) and also varied in terms of the number of included constraints depending on the different feature relationships and the total of derived clauses (around 1,600 SAT clauses in the generated CNF files). Based on these knowledge bases, we randomly generated requirements ($c_i \in C$) that covered 10% of the variables included in the knowledge base. These requirements have been generated so that conflict sets of different cardinalities could be analyzed. We also shuffled C to get different orders because this can affect the number of consistency checks needed.

Completion of a Feature Model Partial Configuration Complexity: The completion of a FM partial configuration is the problem of how to extend that partial configuration into a consistent configuration of the FM. Figure 6 illustrates a partial configuration (grey features {*Debian*} represent selected features), and the Figure 7 shows the complete configuration by the preferred minimal solution (grey features {*Debian*, *texteditor*, *openoffice.org-1*, *openoffice.org-1.2*, *bash*, *gui*, *gnome*}).

A. QuickXPlain *RESULTS*

Table 5 presents a summary of the results of the QuickXPlain performance analysis to identify a preferred minimal conflict of product configurations. In this table, each entry represents the average runtime in *msec* for all knowledge bases with a preferred conflict set of cardinality n (1–16). We can appreciate that the time increases when more conflicts exist in the analyzed product configurations. For the mentioned issue that QuickXPlain identifies only one conflict that, after solving it, a new execution is necessary to identify the remaining one.

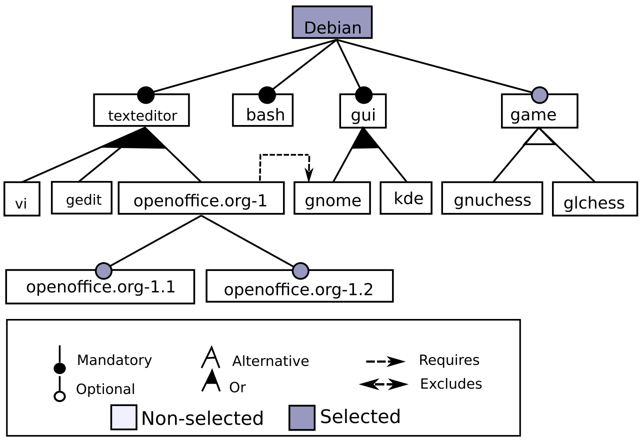


FIGURE 6. Example of a partial product configuration [1].

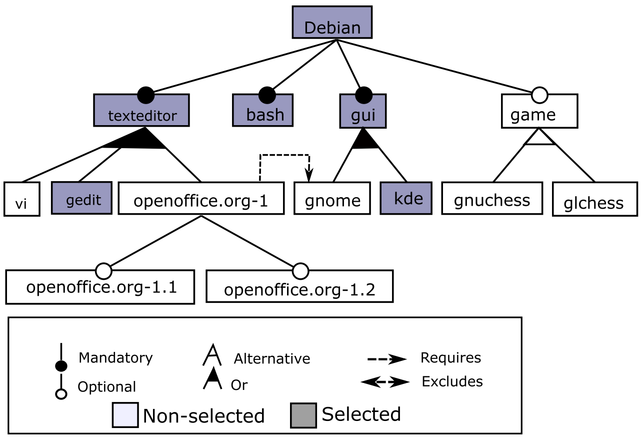


FIGURE 7. Example of completion of partial product configuration of Figure 6 [1].

B. FastDiag *RESULTS*

Table 6 presents a summary of the results of the FastDiag performance analysis to identify a preferred minimal

TABLE 5. **Avg. runtime (in *msec*) of QX when determining minimal conflicts.**

$lmax$	Conflict Cardinality				
	1	2	4	8	16
1	1167,68	2120,68	3415,95	5488,71	9242,53

lmax=1 is equivalent to sequential QuickXPlain. *Each cell follows a heat map colouring: the darker the slower. In bold the cells with faster time for a given conflict cardinality.*

TABLE 6. **Avg. runtime (in *msec*) of FD (lmax=1) for determining preferred diagnosis.**

lmax	Conflict Cardinality				
	1	2	4	8	16
1	1161,14	1162,47	1162,48	1166,44	1166,12

lmax=1 is equivalent to sequential FastDiag. *Each cell follows a heat map colouring: the darker the slower. In bold the cells with faster time for a given conflict cardinality.*

diagnosis of product configurations. In this table, each entry represents the average runtime in *msec* for all knowledge bases with a preferred diagnosis set of cardinality n (1–16). We can appreciate a surprisingly time execution difference between the conflict and diagnosis detection; that is, FastDiag results more efficient than QuickXPlain even though they pursue different tasks. We can appreciate in Table 6 that the time increases when more conflicts exist in the product configurations because FastDiag requires identifying diagnosis of more cardinality.

C. COMPLETION RESULTS

To evaluate the performance of the traditional CSP-based approach for product completion, first, we generate a set of random FMs using the Betty tool-suite [43] to define the number of features and structure of randomly generated FMs. Betty also allows us to define the number of cross-tree constraints in the model. In our experiment, we generate models with the following number of features $|F| = \{50, 100, 500, 1000, 2000, 5000\}$ and the following percentages of cross-tree constraints $c = \{5, 10, 30, 50, 100\}$. For each model, we generate partial configurations with the following percentages of assigned features $a = \{10, 30, 50, 100\}$. We generate 10 random instances for each model and partial configuration. We evaluated the Choco CSP solver [44] for consistency checks. We executed all experiments in an Intel(R) Core (TM) i7-3537U CPU @ 2.00 GHz with 4 GB RAM using a Windows 10 64 bits operating system.

The QuickXPlain, FlexDiag and the product completion code and data for experiments are available in,[1][2] and,[3] respectively.

[1]https://github.com/cvidalmsu/A-Python-QX-implementation

[2]https://github.com/cvidalmsu/A-Python-FD-implementation

[3]https://github.com/cvidalmsu/BOLON-FaMaProdConf-TestSuite

TABLE 7. **Avg. time (in milliseconds) on the completion of partial product configuration of randomly generated Betty FMs by the number of features *n*.**

n	CSP-based approach
50	98.00
100	109.06
500	200.09
1,000	405.89
2,000	1,392.27
5,000	15,677.93
All	2,980.54

V. DISCUSSION

Such as report Vidal-Silva [1], the Conflict detection, diagnosis detection, and product completion represent the most relevant operation for assisting the product completion of VIS. Even though various solutions exist, some of the most relevant and efficient still are QuickXPlain, FastDiag and CSP tools. Just Vidal-Silva et al. [2] present an optimized version of QuickXPlain for parallel computing that represents the future for conflict detection. Likewise, Vidal et al. [45] show the usability and efficiency of applying diagnosis solutions such as FastDiag for product completion.

To show the functionality and evaluate the performance of our solutions, we implemented them using Python and FAMA [46]. Specifically, Python for QuickXPlain and FastDiag, and FAMA and Java for the product completion.

VI. THREATS TO VALIDITY

This work presents relevant operations for the Automated Analysis of Product Configuration of Feature Models. Nonetheless, it is necessary to discuss the following practical issues:

- We implemented our solutions to run in Python and FAMA [46]. For executing QuickXPlain and FastDiag, Python and FAMA should be in the computer. That seems not a problem because Python in 2022 is one of the most used programming environments, and FAMA is freely online accessible.
- We used generated FMs by the use of Betty. Applying product configuration solutions in real models and configuration cases can be more precise. Nonetheless, generated models are adequate for simulation goals.

VII. SPECULATIVE PROGRAMMING

Speculative programming is an optimization technique for writing programs with portions of code that can execute before needing their results. Speculative execution is the pre-execution or pre-calculation of results that can contribute to computing the expected outcome [47]. We can define parallel tasks for executing those pre-calculations for a Thread Level Speculation (TLS) [48]. We can differentiate effective and non-valid speculative executions for which the program execution time can improve or even deteriorate for the cost of orchestrating the parallel tasks, respectively. The current availability of parallel computing resources permits developing solutions using speculative programming

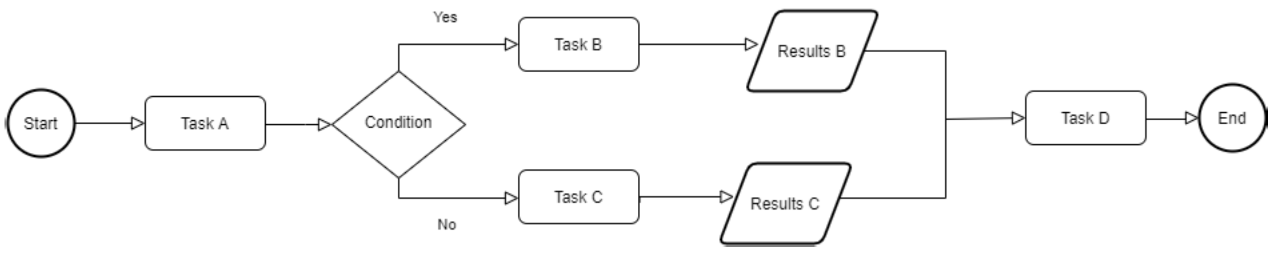


FIGURE 8. Flow diagram example with four tasks *A*, *B*, *C*, and *D* [1].

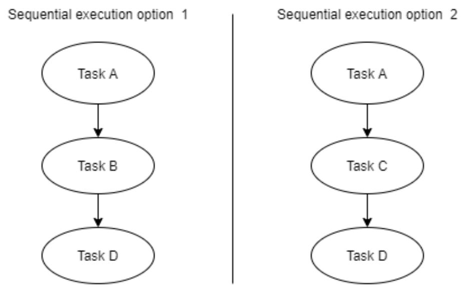


FIGURE 9. Tracking options for the tasks of Figure 8 [1].

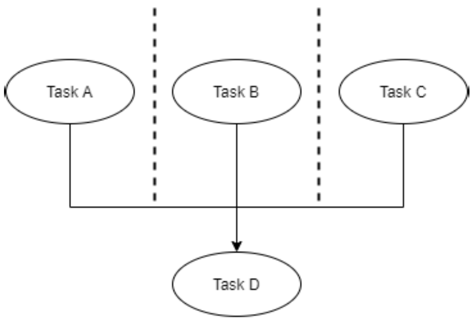


FIGURE 10. Speculative tracking for the tasks of Figure 8 [1].

for eventually getting more efficient solutions [49]. Figure 8 shows a flow diagram with four tasks A, B, C, and D. A is the first task, and after it, task B or C can occur depending on a condition, and then task D would execute before the end. Task D depends of the results of B or C to proceed. Figure 9 shows the tracking for the sequential execution options, each one containing three steps according to the defined sequential order. Figure 10 shows an speculative execution that executes tasks A, B and C in parallel, and then executes task D. Then, task D would proceed after getting the correct input from the previous tasks. In this example, tasks B and C would give an output; then, it is necessary to decide the correct input for D before its execution. Executing this example in a machine with adequate parallel resources would permit the execution more efficient of the speculative solution than the sequential one. This example illustrates that speculating about future actions permits improving the execution time of a set of tasks when resources for parallel computing are able. We can use speculative programming to look for a parallel and more efficient version of existing solutions to solve the first two issues.

The work of Vidal et al. [2] presents the use of speculative programming for conflict detection. Because that work extends QuickXPlain for applying parallel computing and efficiently detecting conflicts, we can apply the approach for diagnosis (one of the ongoing works of this research team). Regarding product completion, a speculative version of FastDiag will be a more efficient solution for that purpose.

VIII. CONCLUSION

This article reviewed the functionality, computing performance, and main details of QuickXPlain, FastDiag and the use of CSP solver that are relevant solutions for conflict detection, diagnosis, and product completion. Additionally, the study provides the base and highlights the speculative programming approach as an algorithmic optimization technique applicable for solutions such as QuickXPlain, FastDiag and diagnosis by completion solutions. This work supports this line of research allowing the appreciation of the relevance of product completion and the operation to follow that purpose.

ACKNOWLEDGMENT

The authors are currently living in a difficult time, plagued by a pandemic. Regardless, they continue working as a research team in the software engineering and product line areas. Their special thanks, first to the University of Seville for giving the fundaments base of this research. Then, thanks to the University of Talca, Talca, Chile, the Universidad Católica del Norte, Antofagasta, Chile, the Universidad Estatal de Milagro, Milagro, Ecuador, and Universidad Continental, Huancayo, Peru, for assisting them during the whole process and providing them with the budget to fulfill their research tasks. They feel encouraged to continue dealing with these research activities in the coming years.

REFERENCES

[1] C. Vidal-Silva, "Configuration analysis for large scale feature models: Towards speculative-based solutionse," Ph.D. dissertation, Dept. de Lenguajes y Sistemas Informáticos, Univ. Seville, Seville, Spain, 2021.
[2] C. Vidal-Silva, A. Felfernig, J. A. Galindo, M. Atas, and D. Benavides, "Explanations for over-constrained problems using QuickXPlain with speculative executions," *J. Intell. Inf. Syst.*, vol. 57, no. 3, pp. 491–508, 2021.
[3] A. Felfernig, G. Friedrich, and D. Jannach, "Conceptual modeling for configuration of mass-customizable products," *Artif. Intell. Eng.*, vol. 15, no. 2, pp. 165–176, Apr. 2001.
[4] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-Based Configuration: From Research to Business Cases*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2014.

[5] Ubuntu. (2019). *16.04.6 LTS (Xenial Xerus)*. accessed: Feb. 1, 2019. [Online]. Available: http://cl.releases.ubuntu.com/16.04/

[6] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Inf. Syst.*, vol. 35, no. 6, pp. 615–636, Sep. 2010, doi: 10.1016/j.is.2010.01.001.

[7] J. A. Galindo, F. Roos-Frantz, D. Benavides, A. Ruiz-Cortés, and J. García-Galán, "Automated analysis of diverse variability models with tool support," in *Proc. 19th Jornadas Ingeniería del Software Y Bases Datos (JISBD)*, Jan. 2014, pp. 160–168.

[8] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon Univ. Softw. Eng. Inst., Pittsburgh, PA, USA, Tech. Rep., CMU/SEI-90-TR-021, Nov. 1990.

[9] S. Apel, D. Batory, C. Kstner, and G. Saake, *Feature-Oriented Software Product Lines: Concepts and Implementation*. Manhattan, NY, USA: Springer, 2013.

[10] D. Batory, "Feature models, grammars, and propositional formulas," in *Proc. 9th Int. Conf. Softw. Product Lines*. Berlin, Germany: Springer-Verlag, 2005, pp. 7–20, doi: 10.1007/11554844_3.

[11] M. Galster, "Variability-intensive software systems: Product lines and beyond," in *Proc. 13th Int. Workshop Variability Model. Softw.-Intensive Syst.*, New York, NY, USA, Feb. 2019, p. 1, doi: 10.1145/3302333.3302336.

[12] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki, "The variability model of the Linux kernel," in *Proc. 4th Int. Workshop Variability Model. Softw.-Intensive Syst.*, vol. 37, D. Benavides, D. S. Batory, and P. Grünbacher, Eds. Linz, Austria: Universität Duisburg-Essen, Jan. 2010, pp. 45–51. [Online]. Available: http://www.vamos-workshop.net/proceedings/VaMoS_2010_Proceedings.pdf

[13] V. Rothberg, N. Dintzner, A. Ziegler, and D. Lohmann, "Feature models in Linux: From symbols to semantics," in *Proc. 10th Int. Workshop Variability Model. Softw.-Intensive Syst. (VaMoS)*, New York, NY, USA, 2016, pp. 65–72, doi: 10.1145/2866614.2866624.

[14] J. Galindo, D. Benavides, and S. Segura, "Debian packages repositories as software product line models. Towards automated analysis," in *Proc. 1st Int. Workshop Automated Configuration Tailoring Appl.*, 2010, pp. 29–34.

[15] J. A. Galindo, H. Turner, D. Benavides, and J. White, "Testing variability-intensive systems using automated analysis: An application to android," *Softw. Quality J.*, vol. 24, no. 2, pp. 365–405, 2016, doi: 10.1007/s11219-014-9258-y.

[16] A. B. Sánchez, S. Segura, J. Parejo, and A. Ruiz-Cortés, "Variability testing in the wild: The Drupal case study," *Softw. Syst. Model.*, vol. 16, no. 1, pp. 1–22, 2015, doi: 10.1007/s10270-015-0459-z.

[17] M. Lienhardt, F. Damiani, E. B. Johnsen, and J. Mauro, "Lazy product discovery in huge configuration spaces," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, New York, NY, USA, Jun. 2020, p. 1509, doi: 10.1145/3377811.3380372.

[18] U. Junker, "QuickXplain: Preferred explanations and relaxations for over-constrained problems," in *Proc. 19th Nat. Conf. Artif. Intell.*, San Jose, CA, USA, 2004, pp. 167–172.

[19] A. Felfernig, D. Benavides, J. Galindo, and F. Reinfrank, "Towards anomaly explanation in feature models," in *Proc. 15th Int. Configuration Workshop*, Aug. 2013, pp. 117–124.

[20] C. Thörn and K. Sandkuhl, *Feature Modeling: Managing Variability in Complex Systems*. Berlin, Germany: Springer, 2009, pp. 129–162, doi: 10.1007/978-3-540-88075-2_6.

[21] R. Mazo, R. Lopez-Herrejon, C. Salinesi, D. Diaz, and A. Egyed, "Conformance checking with constraint logic programming: The case of feature models," in *Proc. IEEE 35th Annu. Comput. Softw. Appl. Conf.*, Aug. 2011, pp. 456–465.

[22] F. Zhou, J. R. Jiao, X. J. Yang, and B. Lei, "Augmenting feature model through customer preference mining by hybrid sentiment analysis," *Exp. Syst. Appl.*, vol. 89, pp. 306–317, Dec. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417417304980

[23] M. Weckesser, M. Lochau, T. Schnabel, B. Richerzhagen, and A. Schürr, "Mind the gap! Automated anomaly detection for potentially unbounded cardinality-based feature models," in *Proc. 19th Int. Conf. Fundam. Approaches Softw. Eng.*, vol. 9633. New York, NY, USA: Springer-Verlag, 2016, pp. 158–175, doi: 10.1007/978-3-662-49665-7_10.

[24] A. R. Santos and E. Santana de Almeida, "Do #ifdef-based variation points realize feature model constraints?" *SIGSOFT Softw. Eng. Notes*, vol. 40, no. 6, pp. 1–5, Nov. 2015, doi: 10.1145/2830719.2830728.

[25] S. Segura, A. B. Sánchez, and A. Ruiz-Cortés, "Automated variability analysis and testing of an E-commerce site: An experience report," in *Proc. 29th ACM/IEEE Int. Conf. Automated Softw. Eng.*, New York, NY, USA, Sep. 2014, pp. 139–150.

[26] M. Nieke, J. Mauro, C. Seidl, T. Thüm, I. C. Yu, and F. Franzke, "Anomaly analyses for feature-model evolution," in *Proc. 17th ACM SIGPLAN Int. Conf. Generative Program., Concepts Exper.*, New York, NY, USA, Nov. 2018, pp. 188–201, doi: 10.1145/3278122.3278123.

[27] A. Khtira, A. Benlarabi, and B. Asri, "Duplication detection when evolving feature models of software product lines," *Information*, vol. 6, no. 4, pp. 592–612, Oct. 2015.

[28] D. M. Le, H. Lee, K. C. Kang, and L. Keun, "Validating consistency between a feature model and its implementation," in *Safe and Secure Software Reuse*, J. Favaro and M. Morisio, Eds. Berlin, Germany: Springer, 2013, pp. 1–16.

[29] A. Gómez and I. Ramos, "Automatic tool support for cardinality-based feature modeling with model constraints for information systems development," in *Information Systems Development, Business Systems and Services: Modeling and Development*. Prague, Czechia: Charles Univ. Prague, Aug. 2010, pp. 271–284, doi: 10.1007/978-1-4419-9790-6_22.

[30] C. Quinton, A. Pleuss, D. L. Berre, L. Duchien, and G. Botterweck, "Consistency checking for the evolution of cardinality-based feature models," in *Proc. 18th Int. Softw. Product Line Conf.*, New York, NY, USA, Sep. 2014, pp. 122–131, doi: 10.1145/2648511.2648524.

[31] A. S. Karataş and H. Oğuztüzün, "Attribute-based variability in feature models," *Requirements Eng.*, vol. 21, no. 2, pp. 185–208, 2016, doi: 10.1007/s00766-014-0216-9.

[32] F. Roos-Frantz, D. Benavides, A. Ruiz-Cortés, A. Heuer, and K. Lauenroth, "Quality-aware analysis in product line engineering with the orthogonal variability model," *Softw. Quality J.*, vol. 20, nos. 3–4, pp. 519–565, Sep. 2012, doi: 10.1007/s11219-011-9156-5.

[33] C. Hwan, P. Kim, and K. Czarnecki, "Synchronizing cardinality-based feature models and their specializations," in *Proc. 1st Eur. Conf. Model Driven Archit., Found. Appl.* Berlin, Germany: Springer-Verlag, 2005, pp. 331–348, doi: 10.1007/11581741_24.

[34] J. A. Galindo, D. Benavides, P. Trinidad, and A.-M. Gutiérrez-Fernández, and A. Ruiz-Cortés, "Automated analysis of feature models: Quo vadis?" *Computing*, vol. 101, no. 5, pp. 387–433, 2019, doi: 10.1007/s00607-018-0646-1.

[35] A. Felfernig, M. Schubert, and C. Zehentner, "An efficient diagnosis algorithm for inconsistent constraint sets," *Artif. Intell. Eng. Des., Anal. Manuf.*, vol. 26, no. 1, pp. 53–62, Feb. 2012, doi: 10.1017/S0890060411000011.

[36] S. Ibraheem and S. Ghoul, "Software evolution: A features variability modeling approach," *J. Softw. Eng.*, vol. 11, no. 1, pp. 12–21, Dec. 2016.

[37] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy, "An empirical study on configuration errors in commercial and open source systems," in *Proc. 23rd ACM Symp. Operating Syst. Princ. (SOSP)*, 2011, pp. 159–172.

[38] Facebook. *More Details on Today's Outage*. Accessed: Jan. 26, 2021. [Online]. Available: https://m.facebook.com/nt/screen/?params=%7B%22note_id%22%3A10158791436142200%7D&path=%2Fnotes%2F%7Bnote_id%7D&_rdr

[39] L. A. Barroso and U. Hoelzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 1st ed. San Rafael, CA, USA: Morgan & Claypool, 2009.

[40] J. Zhanwen-Li, S. He, L. Zhu, X. Xu, M. Fu, L. Bass, A. Liu, and A. B. Tran, "Challenges to error diagnosis in Hadoop ecosystems," in *Proc. 27th Large Installation Syst. Admin. Conf. (LISA)*, 2013, pp. 145–154.

[41] H. Riener and G. Fey, "Exact diagnosis using Boolean satisfiability," in *Proc. 35th Int. Conf. Comput.-Aided Design*, New York, NY, USA, Nov. 2016, pp. 1–8, doi: 10.1145/2966986.2967036.

[42] J. Galindo and D. Benavides, "Towards a new repository for feature model exchange," in *Proc. 23rd Int. Syst. Softw. Product Line Conf. (SPLC)*, C. Cetina, O. Díaz, L. Duchien, M. Huchard, R. Rabiser, C. Salinesi, C. Seidl, X. Tërnava, L. Teixeira, T. Thüm, and T. Zadi, Eds. Paris, France: ACM, 2019, p. 85, doi: 10.1145/3307630.3342405.

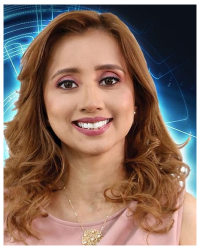
[43] S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, and A. Ruiz-Cortés, "BeTTy: Benchmarking and testing on the automated analysis of feature models," in *Proc. 6th Int. Workshop Variability Model. Softw.-Intensive Syst.*, 2012, pp. 63–71, doi: 10.1145/2110147.2110155.

[44] D. Benavides, S. Segura, P. Trinidad, and A. R. Cortés, "FAMA: Tooling a framework for the automated analysis of feature models," in *Proc. 1st Int. Workshop Variability Model. Softw.-Intensive Syst. (VAMOS)*, 2007, pp. 129–134.

[45] C. Vidal, J. A. Galindo, J. Giráldez, and D. Benavides, "Automated completion of partial configurations as a diagnosis task. Using FastDiag to improve performance," in *Proc. ISMIS*. Graz, Austria: Graz Univ. Technology, Sep. 2020, pp. 107–117. [Online]. Available: https://ismis.ist.tugraz.at/industry-session/
[46] *How to Prevent and Fix Package Dependency Errors in Ubuntu*. Accessed: Feb. 10, 2020. [Online]. Available: https://www.isa.us.es/fama/
[47] J. Tatemura, "Speculative parallelism of intelligent interactive systems," in *Proc. 21st Annu. Conf. IEEE Ind. Electron. (IECON)*, Nov. 1995, pp. 193–198.
[48] J. F. Martínez and J. Torrellas, "Speculative synchronization: Applying thread-level speculation to explicitly parallel applications," in *Proc. 10th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, New York, NY, USA, 2002, p. 18, doi: 10.1145/605397.605400.
[49] B. Bramas, "Increasing the degree of parallelism using speculative execution in task-based runtime systems," *PeerJ Comput. Sci.*, vol. 5, p. e183, Mar. 2019.

CRISTIAN VIDAL-SILVA received the B.S. degree in information engineering from the Faculty of Engineering, Catholic University of Maule, Talca, Chile, in 2003, the M.S. degree in computer science from the University of Concepción, Concepción, Chile, in 2007, the M.S. degree in computer science from Michigan State University, East-Lansing, MI, USA, and the Ph.D. degree in software engineering from the University of Seville, Seville, Spain.

From 2004 to 2007, he was a Research Assistant with the Engineering Faculty, University of Concepción, Concepción, Chile, and a part-time Professor at the Catholic University of Maule. From 2008 to 2013, he was a Professor at the Faculty of Economics and Business, University of Talca, Talca. From 2014 to 2018, he worked as a Professor and a Researcher in software engineering and computer science at the University of Playa Ancha, the University of Viña del Mar, and the Autónoma University of Chile, Chile. From 2019 to 2021, he worked as a Professor at the Administration Department, Catholic University of the North, Antofagasta, Chile. He is currently a full-time Professor and a Researcher with the Videogame Development and Virtual Reality Engineering (VDVRE) School, Engineering Faculty, University of Talca. He is the main author of more than 30 articles in research areas, such as feature-oriented and aspect-oriented software engineering, machine learning, and digital circuits and programming. He worked as a Reviewer of WOS and Scopus journals. He is a Fulbright Scholar.



JESENNIA CARDENAS-COBO received the B.S. degree in information systems from the Escuela Superior Politecnica del Litoral (ESPOL), Ecuador, the Diploma degree in higher education by competences from the Technical University of Ambato, Ecuador, and the M.Sc. degree in business administration from the Business and Technological University of Guayaquil, Ecuador. She is currently pursuing the Ph.D. degree in software engineering with the University of Seville, Spain. She is also a full-time Professor and the Dean of the Faculty of Engineering Sciences, State University of Milagro (now UNEMI). She has more than 20 years of professional experience in higher education. Her main research interests include software products and arti?cial intelligence applied to engineering education.

AURORA SÁNCHEZ ORTIZ received the M.Sc. degree in industrial engineering from the Universidad de Chile, Chile, and the Ph.D. degree in information science with a minor in management information systems from the University of North Texas, Denton, TX, USA, in 2003. She is currently an Associate Professor of information systems and the Director of the Information Technology Management Research Center, Business School, Universidad Católica del Norte, Chile, and the "Collaboration Network in Information Technology and Systems in Chile" (RedSTI). She has been published in journals, such as *Sustainability*, *Journal of Theoretical and Applied Electronic Commerce Research*, and *Formación Universitaria*. She also has academic articles in Springer books and conferences, such as AMCIS, Academy of Management, and Conf-IRM. Her research interests include e-government, smart cities, and digital transformation. She received the Fulbright Scholarship for her Ph.D. degree.

VANNESSA DUARTE received the degree in computer science engineering from the Universidad Nacional Experimental del Táchira, Venezuela, and the D.Eng. degree from the Universidad Central de Venezuela. She has ten years of experience in university education. She is currently working as a Professor and a Researcher at the School of Business Sciences, Universidad Católica del Norte, Coquimbo, Chile. Her research interest includes bioengineering, mainly in the development of biological models of bone remodeling with mechanical and piezoelectric loads, and data science. Her particular attention to the implementation of machine learning applied to different areas, such as health, business, robotics, and decision making.

MIGUEL TUPAC-YUPANQUI received the master's degree in systems engineering with a major in business systems management from the National University of the Center of Peru. He is currently a full-time Teacher with the Continental University, Peru. He is also an Electronics Engineer with Ricardo Palma University, Peru. Since 2005, he has been the Academic Director in charge of the Professional Academic School (EAP) of Systems Engineering and Informatics, Continental University, contributing with academic and administrative management. He has contributed to teaching and research.