Multiple software product lines to configure applications of internet of things

Guadalupe-Isaura Trujillo-Tzanahua, Ulises Juárez-Martínez, Alberto-Alfonso Aguilar-Lasserre, María-Karen Cortés-Verdín, Catherine Azzaro-Pantel

Multiple software product lines to configure applications of internet of things

Guadalupe-Isaura Trujillo-Tzanahua[1] ✉, Ulises Juárez-Martínez[1], Alberto-Alfonso Aguilar-Lasserre[1], María-Karen Cortés-Verdín[2], Catherine Azzaro-Pantel[3]

[1]Division of Research and Postgraduate Studies, Tecnológico Nacional de México/I.T. Orizaba, Orizaba, Veracruz México

[2]Facultad de Estadística e Informática. Universidad Veracruzana, Xalapa, Veracruz, México

[3]Université de Toulouse, Laboratoire de Génie Chimique, LGC UMR 5503 CNRS INP-ENSIACET, 4, Allé Emile Monso BP 84234 F-31432 Toulouse, France

✉ *E-mail: gtrujillot@ito-depi.edu.mx*

Abstract: Software product lines (SPL) emulate the industrial production lines that are capable of generating large volumes of products through reuse schemes and mass production. A multi product line (MPL) aims to reuse of several SPL. Feature models are often used to manage the existing resources of SPLs and define valid products through notations and relationships such as mandatory, optional, and alternative features. The main contribution of this study is a method to manage the variability of multiple SPL and generate a new portfolio of products for Internet of Things (IoT). For this, the problem of developing a universal feature model (FM) for an MPL from merging the FMs of the individual SPLs with a Search-Based Software Engineering (SBSE) technique is addressed. In addition, the authors propose a multi-objective optimisation model to maximise the reusability and compatibility between features and minimise the development cost. The model facilitates the design of an MPL-feature model. Authors' empirical results show that the proposed model solved by genetic algorithms allows to configure a variety of software products and to determine the scope of the MPL.

1 Introduction

Software engineering incorporates practices or methodologies such as techniques of optimisation and product lines used in other already consolidated industries as viable alternatives for mass customisation, cost reduction, and efficient use of resources. This is in order for companies to meet the changing requirements of customers and get a competitive advantage in today's markets.

Software Product Lines (SPL) emulate industrial product lines that are capable of generating large volumes of products through reuse and optimisation of inputs and processes. A product line is a family of products developed from a set of variables and similar features. When the final product is software, it is called SPL. Unlike conventional software development paradigms that focus on developing unique systems, SPL Engineering (SPLE) is a paradigm focused on building a family of software systems across reuse and development assets in a given domain. This paradigm is used by companies such as Siemens, Nokia, Toshiba, Bosch, HP, Philips among others.

Multi product lines (MPL) emerge as a paradigm of flexible development that helps companies improve their products and keep them updated through the reuse of software artefact and products of several heterogeneous SPL.

On the other hand, the search-based software engineering (SBSE) is an area in which the problems of software engineering are reformulated and modelled as optimisation problems and subsequently, they are solved using techniques, algorithms, and search strategies. These problems are characterised by a large search space, in which multiple objectives and constraints are involved and may be conflicting. This search aims to identify, among all possible solutions, one that is good enough according to the appropriate metrics. There are many studies in the literature that have explored the application of techniques of SBSE as simulated annealing (SA), genetic algorithms, ant colony optimisation and particle swarm optimisation in various activities of the life cycle of the SPLE for example optimal feature selection [1–6], test generation [7, 8], architecture [9, 10], and among others. These capabilities and applications are attractive to meet some challenges that may arise when implementing an MPL such as SPL reuse, interoperability, product derivation, reference architecture, among others [11].

MPL needs to manage SPL efficiently to identify the variation points and the level of reuse and so configure new products in different domains. To generate products in an MPL and reuse artefacts from the SPLs available; it is advisable to merge their FMs and determine the scope which will be an MPL [12]. A MPL's scope is a description of the products that constitute the MPL or the production capacity of an MPL from *N* SPL. Within that scope, the disciplined reuse of core assets, such as requirements, features, designs, test cases, tools, and other software development artefacts, greatly reduces the cost of product development.

In this paper, we propose a SBSE method to manage the variability (i.e. the ability to change, customise, or configure systems) of multiple SPL. First, the problem of developing a universal FM of the MPL from merging the FMs of the individual SPLs with a SBSE is addressed. Then, we propose a multi-objective optimisation model to maximise the reuse and compatibility between features and minimise the development cost. The model facilitates the design of an MPL-FM that integrates all the features, which allows generating a new portfolio of products for the Internet of Things (IoT).

IoT is a concept that refers to the digital interconnection of everyday objects with the Internet and provides them with intelligence, so they automate tasks and make life a little easier, it reduces human effort and improve their quality of life. Currently, there are many applications of IoT such as home automation, healthcare, smart greenhouse, education, entertainment, social life, energy conservation, among others. These multi-domain IoT applications are interesting for an MPL implementation since this paradigm arises for the development of large and complex systems through several independent SPLs that are developed by several organisations [13, 14] with different hardware and software technologies for different geographical areas and in any context.

Our empirical results show that the proposed model solved by genetic algorithms allows to configure a variety of software

Table 1 SBSE techniques for SPLs

Problem domain	SA	GA	ACO	Pairwise	BA	MOO
feature selection/product configuration	Yes [26]	Yes [1, 27, 28, 29]	Yes [2]	—	—	Yes [27]
architecture	—	—	—	—	—	Yes [9, 10]
maintenance and evolution/feature location	—	Yes [30]	—	—	—	—
SPL testing/multi-objective test generation	—	Yes [8, 31]	—	Yes [7]	Yes [32]	Yes [8]
product platform scope	Yes [33]	—	—	—	—	—

products and to determine the scope of the MPL. The paper is organised as follows. Section 2 presents the background on SPL, MPL, and SBSE. Section 3 describes the problem of MPL-feature model as a search problem and the proposed multi-objective optimisation model. Section 4 describes a method to generate the MPL-feature model using a multi-objective optimisation model. Section 5 is presents the as case study of the configuration of applications to IoT reusing two SPL for smart building automation. Section 6 presents the empirical results of experiments to evaluate the algorithms. Finally, Section 7 presents the conclusion and future work.

2 Background

SPL and SBSE are the two fields that constitute the essence of this research. This section provides the necessary background information on these topics.

2.1 Software product lines

A SPL is defined as 'a set of software-intensive systems that share a common, managed set of features to satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [15]'.

SPLE separates two processes: Domain Engineering and Application Engineering [16]. Domain Engineering is the process responsible for establishing the platform of reusable assets, specifically defining what is common and variable in the SPL. Therefore, Application Engineering also called product derivation is the process in which applications are constructed by reusing domain artefacts and exploiting the variability of the SPL. The main activity of the derivation of products is to find the best configuration of features to meet the customer requirements. These requirements can be functional and non-functional; some competing and even entering conflict each other.

Variability is defined as the ability of a software system or artefact to be efficiently extended, changed, customised, or configured for use in a particular context [17]. The variability aims to maximise the return on investment (ROI) to build and maintain products during a specific period of time. Variability is explicated in a software variability model called FM which distinguishes different applications of SPL.

A feature model (FM) graphically or textually represents an SPL through possible combinations of features. A FM is used to capture common and variable aspects of all products in a SPL in terms of features. This model also guides the product configuration in an SPL. FMs are usually used to specify sets of mandatory, optional, and alternative features which define valid products in the SPL. A feature is a relevant functionality for stakeholders because it captures common and variable aspects of the systems in an SPL [18].

2.2 Multi product lines

For many years researchers have tried to reuse several artefacts of multiple SPL [19–24]. An MPL (MPL or MSPL) is a special type of SPL that reuses most of the software artefacts of various SPLs, which are developed, managed, and implemented independently. According to Holl [24], an MPL is 'a set of several self-contained but still interdependent product lines that together represent a large-scale or ultra-large scale system'. Acher [22, 23] defined an MPL as 'an SPL that manages a set of SPLs {SPL_1, SPL_2, …, SPL_n}'. Such set of products is described by a FM called MPL-FM (FM_{MPL}) [25] that contains all the features of the SPLs and their interrelationships.

2.3 SBSE techniques for SPLs

Over the last decade, the scope of SBSE has expanded to cover different domains of SPL life cycle, such as feature selection, test generation, architecture, maintenance, among others. Table 1 shows the optimisation problems identified in the SPLs and SBSE techniques used to solve them.

The selection of search-based techniques is performed keeping in view the way they can solve most of the issues lying in the SPLE domain. SBSE techniques and how they have been used to resolve different SPL issues are described below.

SA is named so because it was inspired by the physical process of annealing, the cooling of a material in a heat bath. When a solid material is heated past its melting point and then cooled back into its solid state, the structural properties of the final material will vary depending on the rate of cooling. SA is an approximation algorithm to the optimal solution and is used in the search for solutions to large optimisation problems. In [33], this technique is used to optimise the scope of a software product platform.

Genetic algorithms (GA) are techniques used to solve search and optimisation problems that simulate the natural evolution process. The algorithms are based on Darwin's theory in which the most apt individuals survive while the less-adapted individuals tend to disappear. The search procedure aims to maintain a population of potential solutions (chromosomes) while conducting parallel research for solutions with a high fitness function. GAs have several variations, specifically for genetic operators (crossover, mutation), selection, and how individuals are replaced to form the new population. In [30], the authors present an approach to feature location in model-based SPLs that is called genetic algorithm to feature location (GA-FL). FL is known as the process of finding the set of software artefacts that realise a particular feature.

Ant colony optimisation (ACO) is a technique for approximate optimisation to solve computational problems that search the best paths or routes in graphs. The inspiring source of ACO algorithms are real ant colonies. Wang and Pang presented a polynomial algorithm for feature selection with constraints in SPL using ACO [2]. The authors propose a method to transform a FM from and/or tree to multistage-directed graphs. Then, the feature selection problem is converted to find an optimal path from the origin to the destination.

Pairwise SPL testing aims to select a set of products (test suite), such that their combined coverage contains all the possible pairwise feature combinations of the SPL.

Bat algorithm (BA) is one of the recently proposed heuristic algorithms imitating the echolocation behaviour of bats to perform global optimisation. In [32], an approach called SPLBA is presented for the reduction of SPL tests using a bat-inspired algorithm.

Multi-objective optimisation (MOO) also called multi-criteria optimisation considers optimisation problems involving more than one objective function to be optimised simultaneously. A product line architecture design is an optimisation problem that has been effectively solved using multi-objective algorithms to find optimal solutions that satisfy defined objectives. For example, MOA4PLA (Multi-Objective Optimisation Approach for PLA design) is a search-based approach to support the product line architectures [34]. MOA4PLA uses a PLA modelled in a UML class diagram and optimise it. A set of solutions with the best trade-off between

Table 2 Notation

Notation	Description
f	feature
n	number of features
m	number of SPLs. variable equal to the number of feature models
$\mathrm{i} = 1, 2, \ldots, n$	index of a feature
$j = 1, 2, \ldots, m$	index of a SPL
x_{ij}	variable equal to 1 if the feature i is assigned to SPL j; 0 otherwise
r_{ij}	reusability of feature i if it is assigned to SPL j
w_{ij}	development cost of the feature i if it is assigned to SPL j
p_{ij}	profit or score of the feature i if it is assigned to SPL j for compatibility
W	the user's available budget
R	limit value of reusability required by the user
MA	set of mandatory features
O	set of optional features
XOR	set of all exclusive alternatives features
OR	set of all non-exclusive alternatives features

objectives is generated and the architect may select which architecture will be used. In [8], an approach is presented to handle multiple conflicting objectives in test generation for SPLs. This approach combines genetic algorithms and constraint solving techniques to deal with the following objectives: (i) maximising the pairwise coverage, (ii) minimising the number of products selected, and (iii) minimising the overall test suite cost.

3 Reformulating SPLE as a search problem

For product derivation, first of all, it is advisable to design a universal FM (MPL-FM) that merges FMs of the individual SPLs with a SBSE technique. In order to reformulate this issue of SPLE as a search problem, it is necessary to define:

- A representation of the problem which is amenable to symbolic manipulation.
- A definition of fitness function to quantify the optimality of a solution so that particular solution may be ranked against all the other solutions.

3.1 Problem formulation

In order to solve the MPL-FM problem in multiple SPL, it is necessary to define the problem. In this section, we first define the MPL-FM problem and then proceed to model it as a search problem. To formulate this problem mathematically, the notations are introduced in Table 2.

Given m features models and a requirements specification, a configuration is a set of features to select and that they will integrate the MPL. These features should be compatible between itself and the configuration should meet the requirements of the application and optimise reuse of SPL artefacts.

Thus, if a particular software product is defined by n features f, with each feature $x_{\mathrm{ij}} \in N_{\mathrm{i}}$ having a value by reusability of r_{ij}, a selection reward p_{ij} by compatibility and a development cost w_{ij}, it is possible to define mathematically the MPL-FM problem as a multi-objective optimisation model.

Reusability R_{ij} (1) defines the frequency of usage U_{ij} (2) of a feature i in a FM j.

$$R_{\mathrm{ij}} = \sum_{\mathrm{i}=1}^{n} \frac{U_{\mathrm{ij}}}{n} \tag{1}$$

$$U_{\mathrm{ij}} = \sum_{\mathrm{i}=1}^{n} U_{\mathrm{ij}} \tag{2}$$

3.2 Multi-objective optimisation model

The multi-objective optimisation model (3) simultaneously integrates and optimises three objective functions: software reusability ($Z1$), software compatibility ($Z2$), and cost ($Z3$) with the same set of features.

The objective function (4) is an optimisation by maximisation, which aims to select a feature from every SPL with the maximum total benefit (reusability) while the development cost of the chosen features must not exceed the budget restriction W (7).

The objective function (5) is an optimisation by maximisation, which aims to select a feature from every SPL with the maximum compatibility while the development cost of the chosen features must not exceed the budget restriction W (7).

The objective function (6) is an optimisation by minimisation, which aims to select a feature from every SPL with the minimum development cost while the reusability should exceed the limit value of reusability (8).

$$\text{Maximise } Z_1, Z_2 \text{ and Minimise } Z_3 \tag{3}$$

$$\text{Maximise } Z_1 = \sum_{\mathrm{i}=1}^{m} \sum_{j=1}^{n} x_{\mathrm{ij}} r_{\mathrm{ij}} \tag{4}$$

$$\text{Maximise } Z_2 = \sum_{\mathrm{i}=1}^{m} \sum_{j=1}^{n} x_{\mathrm{ij}} p_{\mathrm{ij}} \tag{5}$$

$$\text{Minimise } Z_3 = \sum_{\mathrm{i}=1}^{m} \sum_{j=1}^{n} x_{i\mathrm{j}} w_{\mathrm{ij}} \tag{6}$$

subject to

$$\sum_{\mathrm{i}=1}^{m} \sum_{j \in N_i} w_{\mathrm{ij}} x_{\mathrm{ij}} \leq W, \tag{7}$$

$$\sum_{\mathrm{i}=1}^{m} \sum_{j \in N_{\mathrm{i}}} r_{\mathrm{ij}} x_{\mathrm{ij}} \geq R, \tag{8}$$

$$\sum_{j \in N_{\mathrm{i}}} x_{\mathrm{ij}} = 1, \quad \forall \mathrm{i} \in \{1, 2, \ldots, n\} \tag{9}$$

$$x_{\mathrm{ij}} \in \{0, 1\}, \quad \forall \mathrm{i} = \{1, 2, \ldots, n\} \quad \forall j = \{1, 2, \ldots, m\}, x_{\mathrm{ij}} \in N_{\mathrm{i}} \tag{10}$$

$$\mathrm{xp} = 1 \rightarrow x_{\mathrm{ij}} = 1 \quad \mathrm{i} \in \mathrm{MA} \tag{11}$$

$$x_p = 0 \rightarrow x_{\mathrm{ij}} = 0 \quad \mathrm{i} \in O \tag{12}$$

```
Generate the initial population;
Evaluate fitness;
while termination condition is not true do
    Selection;
    Crossover;
    Mutation;
    Evaluate fitness;
end
```

Fig. 1 *Algorithm 1: Genetic algorithm*

Table 3 Algorithms

Algorithm	Variable type	Description
NSGA-II	continuous	elitist multi-objective genetic algorithm
NSGA-IIb	continuous	elitist multi-objective genetic algorithm
NSGA-II MI	continuous, integer	mixed elitist multi-objective genetic algorithm
NSGA-II MIB	continuous, integer, binary	mixed elitist multi-objective genetic algorithm
MOGA	binary	multi-objective genetic algorithm with binary representation

Fig. 2 *Representation of chromosome*

$$x_p = 1 \rightarrow \sum_{i \in P} x_{ij} = 1 \quad \forall P \in \mathrm{XOR}_p \tag{13}$$

$$x_p = 1 \rightarrow \sum_{i \in Q} x_{ij} \geq 1 \quad \forall Q \in \mathrm{OR}_p \tag{14}$$

Constraint (9) ensures selecting a single feature from each j SPL.

Constraint (10) ensures each feature $f_{ij} \in N$ having an associated decision variable $x_{ij} \in \{0, 1\}$, such that $x_{ij} = 1$ if feature i is assigned to SPL j and it will be included in the final software product. Otherwise $x_{ij} = 0$, if this feature is not included in the product.

Constraint (11) ensures that the mandatory feature ij ∈ MA is selected if its parent is selected.

Constraint (12) ensures that the optional feature ij ∈ O is not selected if its parent is not selected.

Constraint (13) ensures that for each XOR feature set, exactly one feature in P is selected if p is selected.

Constraint (14) ensures that for each OR feature set, at least one feature in Q is selected if p is selected.

3.3 Solution procedure

The MPL-FM problem is identified as a non-deterministic polynomial-time hard problem (NP-Hard). In this paper, genetic algorithms are used to find the best result for the MPL FM problem because they are capable to solve many optimisation problems due their good performance to deal with combinatorial explosion effectively. In addition, the genetic algorithms have also been explored in various aspects of SPL to solve problems such as feature selection, testing, variability management, feature location in model-based SPL, among others.

Algorithm 1 (See Fig. 1) shows the general process of genetic algorithm.

The multi-objective optimisation model was evaluated using various genetic algorithms included in the MULTIGEN library [35]. This library was developed by the Process Optimisation Department of the Chemical Engineering Laboratory of the Institute National Polytechnique Toulouse (INP). MULTIGEN includes eight algorithms (Table 3), which are distinguished by their structure and type of variables (continuous, integer, or binary).

The representation of a chromosome used in genetic algorithm is illustrated by an example as show in Fig. 2.

4 Generating MPL-feature model

The proposed method for managing MPL variability and generating a new product portfolio for IoT is presented in Fig. 3. This method is based on the analysis and comparison of FMs to find matches between the features maximising compatibility and reusability of features at the lowest possible cost. The proposed method has six main steps:

(i) Identification of variability. As a first step, it is necessary to identify the variability available in the SPL, which consists of listing common and variable features of the products and their relationships (mandatory, optional and alternative) for each SPL.

(ii) Evaluate the SPLs that will integrate the MPL. This step refers to evaluate the reusability and compatibility of the features from every SPL that will integrate the MPL. In this step, the FMs are translated into the data required by the model and the necessary calculations are performed as r_{ij}, p_{ij}, and c_{ij}.

(iii) Implementation of the multi-objective optimisation model proposed in MULTIGEN.

(iv) Solve the multi-objective optimisation model with the genetic algorithms included in MULTIGEN.

(v) Compare the results obtained by each genetic algorithm and choose the optimal configuration.

(vi) Create MPL-FM. With the results obtained from the multi-objective optimisation model, create a new FM (MPL-FM). Merge features with high level of reuse through compositional techniques.

5 Case study

We validate our proposed multi-objective optimisation model based on experiments made on a case study consisting on a family of n product variants (Table 4) for IoT applications. The case study involves the development of an MPL for manufacturing IoT applications in various automation domains such as home, hotels, smart greenhouse, offices, among others.

The MPL must reuse two SPL previously developed by different development teams and distinguished mainly by technology used such as programming languages, modelling notations, and hardware. The two SPLs are capable of generating n software products for smart building automation and are represented by FMs (see Figs. 4 and 5) respectively.

The problem is how to implement a configuration tool for multiple SPL to develop IoT applications. This tool should concentrate the good configurations (i.e. a combination of features or products) that meet the compatibility and reusability requirements and with the lowest possible development cost.

As shown in Fig. 4, the mandatory features lighting, hardware, or sensor must be selected if any product is implemented.

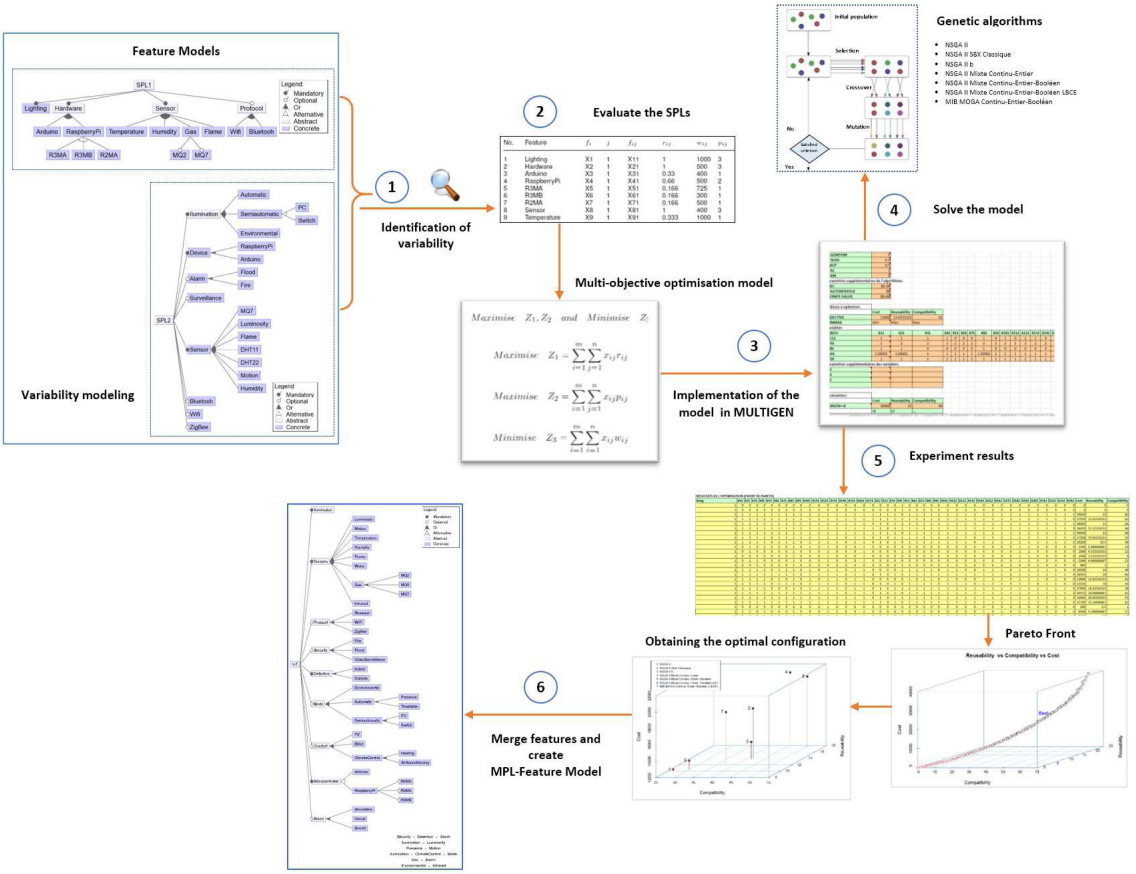

Fig. 3 *Proposed method*

Examples of optional features are Protocol, MQ2, or MQ7. The FM shows alternative features which may be exclusive (XOR) or non-exclusive (OR). An exclusive feature implies that only a sub-feature can be selected from the alternative features such as RaspberryPi. OR hierarchical relationships such as hardware, sensor, and protocol allow choosing more than one option for the product.

6 Computational results

This section presents empirical results from experiments we performed to evaluate the algorithms.

6.1 Dataset

A sample of the dataset for features to generate a portfolio of products for IoT is provided in Table 4.

6.2 Parameter setting

Table 5 shows the input parameters used for the execution of each genetic algorithm. These parameters can influence the population diversity. For example, an excessively high crossover rate will cause the solution to converge quickly before the optimum is found. On the other hand, a low crossover rate decreases the population diversity and results in long computation time.

The mutation rate also influences the GA performance, as it determines the frequency of random search. Generally, a very low mutation rate is recommended to avoid that the genetic algorithm process becomes a pure random search, which impairs the property of GA. The population size may be the most distinct factor influencing the population diversity.

6.3 Experiment results

The algorithms were implemented in MULTIGEN and tested on a PC with an Intel(R) Core(TM) i7 2.8 GHz processor, 16 GB RAM and the Microsoft Windows 10 operating system. In order to measure the performance of the multi-objective optimisation model, it is necessary to identify the best approach for identifying candidate solutions. We evaluate all genetic algorithms available in the MULTIGEN library using the following metrics: desirability value for reusability, compatibility, cost, and time complexity.

The proposed multi-objective optimisation model was executed a total of eight times under the same parameters. For each execution, the initial values of the optimisation variables were changed in order to start the search from different points and to observe if the evolution behaves in the same way. The eight executions of the model gave results greater than those required; therefore, it is possible to conclude that the model performs adequately in the proposed optimisation scenario.

We propose to find the configuration of features that would allow to generate a software product that costs less than $ 30,000.00 USD, a minimum reusability of 15 and a minimum compatibility of 40.

Fig. 6 shows an example of the Pareto Fronts obtained when testing each algorithm to solve the multi-objective optimisation model. The results are indicated with a 1 if the feature was selected and with a 0 if it was not selected.

In order to better visualise the optimal results obtained by all genetic algorithms implemented in MULTIGEN, we present the results of the multi-objective optimisation are plotted in three-dimensional graphs.

The Pareto Fronts show the reusability-compatibility-cost relationship in the feature configuration of an MPL using the following algorithms such as NSGA II (Fig. 7), NSGA II SBX Classique (Fig. 8), NSGA II b (Fig. 9), NSGA II Mixte Continu-Entier (Fig. 10), NSGA II Mixte Continu-Entier-Booléen (Fig. 11), NSGA II Mixte Continu-Entier-Booléen LBCE (Fig. 12) and MIB MOGA Continu-Entier-Booléen (Fig. 13).

6.4 Discussion of optimisation results

We consider a scenario in which the balance between the technical criteria (reusability, compatibility) and economic (cost) have the

Table 4 Test problem details

No.	Feature	f_i	j	f_{ij}	r_{ij}	w_{ij}	p_{ij}	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6
1	lighting	X1	1	X11	1	1000	3	✓	✓	✓	✓	✓	✓
2	hardware	X2	1	X21	1	500	3	✓	✓	✓	✓	✓	✓
3	Arduino	X3	1	X31	0.33	400	1	✓	—	—	—	—	✓
4	RaspberryPi	X4	1	X41	0.66	500	2	—	✓	✓	✓	✓	—
5	R3MA	X5	1	X51	0.166	725	1	—	✓	—	—	—	—
6	R3MB	X6	1	X61	0.166	300	1	—	—	✓	—	—	—
7	R2MA	X7	1	X71	0.166	500	1	—	—	—	✓	—	—
8	sensor	X8	1	X81	1	400	3	✓	✓	✓	✓	✓	✓
9	temperature	X9	1	X91	0.333	1000	1	✓	—	—	—	—	✓
10	humidity	X10	1	X101	0.666	800	2	✓	✓	✓	—	—	✓
11	gas	X11	1	X111	0.166	500	1	✓	—	—	—	—	—
12	MQ2	X12	1	X121	0.5	200	1	✓	✓	—	—	—	✓
13	MQ7	X13	1	X131	0.166	300	1	✓	—	—	—	—	—
14	flame	X14	1	X141	0.166	1650	1	✓	—	—	—	—	—
15	protocol	X15	1	X151	0.166	320	1	✓	—	—	—	—	—
16	Wifi	X16	1	X161	0.666	298	2	✓	✓	✓	—	✓	—
17	bluetooth	X17	1	X171	0.5	2356	1	—	✓	—	✓	—	✓
18	illumination	X1	2	X12	1	2250	3	✓	✓	✓	✓	✓	✓
19	automatic	X2	2	X22	0.5	3500	1	✓	✓	—	✓	—	—
20	semiautomatic	X3	2	X32	0.666	1500	2	✓	—	✓	—	✓	✓
21	environmental	X4	2	X42	0.333	2000	1	✓	—	✓	—	—	—
22	computer	X5	2	X52	0.666	1000	2	✓	✓	✓	—	✓	—
23	switch	X6	2	X62	1	2000	3	✓	✓	✓	✓	✓	✓
24	device	X7	2	X72	1	1000	3	✓	✓	✓	✓	✓	✓
25	RaspberryPi	X8	2	X82	0.666	900	2	—	—	✓	✓	✓	✓
26	Arduino	X9	2	X92	0.333	800	1	✓	✓	—	—	—	—
27	alarm	X10	2	X102	0.666	1000	2	✓	—	✓	—	✓	✓
28	flood	X11	2	X112	0.5	860	1	✓	—	✓	—	✓	—
29	fire	X12	2	X122	0.5	950	1	✓	—	✓	—	—	✓
30	surveillance	X13	2	X132	0.666	2500	2	✓	✓	✓	✓	—	—
31	sensor	X14	2	X142	1	1000	3	✓	✓	✓	✓	✓	✓
32	MQ7	X15	2	X152	0.333	500	1	✓	—	✓	—	—	—
33	luminosity	X16	2	X162	1	1200	3	✓	✓	✓	✓	✓	✓
34	flame	X17	2	X172	0.333	980	1	✓	✓	—	—	—	—
35	DHT11	X18	2	X182	0.5	250	1	✓	—	✓	—	✓	—
36	DHT22	X19	2	X192	0.666	300	2	—	✓	✓	✓	—	✓
37	motion	X20	2	X202	0.333	700	1	✓	—	✓	—	—	—
38	humidity	X21	2	X212	0.5	800	1	✓	—	✓	—	✓	—
39	bluetooth	X22	2	X222	0.666	1000	2	✓	✓	✓	—	—	✓
40	Wifi	X23	2	X232	0.5	1200	1	✓	—	—	✓	—	✓
41	ZigBee	X24	2	X242	0.166	2000	1	—	—	—	—	✓	—

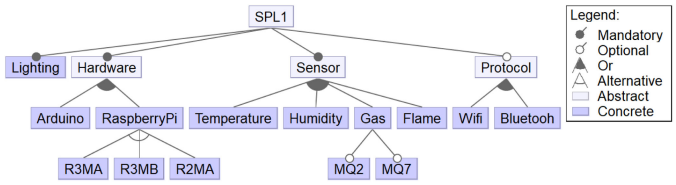

Fig. 4 *Feature Model of SPL1*

same importance when we choose possible configurations of features and, therefore, we decide the product generation.

After analysing the execution results of NSGA II and MOGA algorithms (Fig. 14), we determined that the best alternative to the proposed optimisation problem is the implementation of NSGA II Mixte Continu-Entier-Booléen, both in execution time (1060 s) and in the results obtained (Table 6).

In our tests, the obtained configuration satisfied the expected conditions (subsection 6.3) and we obtained a development cost of $20,728 USD with values higher than expected reusability (17.33) and compatibility (51).

6.5 MPL-feature model

In the proposed method, the mechanism for generating a universal FM for MPLs is mainly based on the optimal configuration obtained by the multi-objective optimisation model (Table 7).

Fig. 15 shows the MPL-FM for IoT applications. The rectangles indicate features and the lines between them are their relations.

7 Conclusion

We present a method formalisation for feature modelling for an MPL. This method allows to develop a universal FM of the MPL from merging the FMs of the individual SPLs using genetic

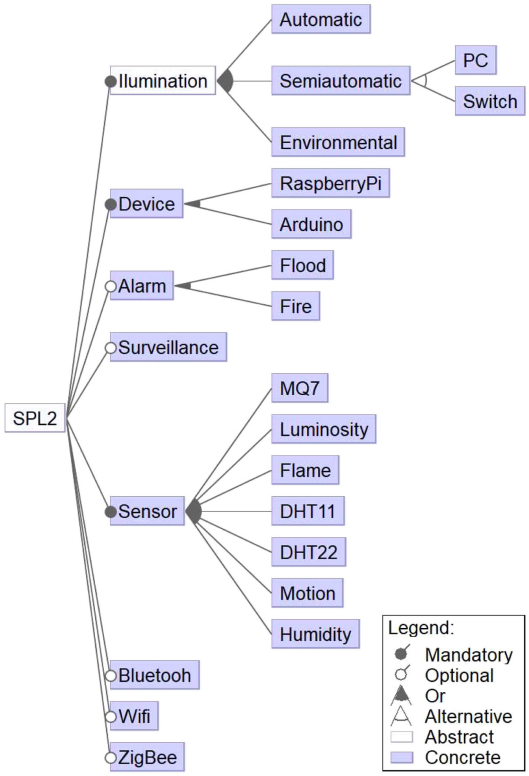

Fig. 5 *Feature Model of SPL2*

Table 5 Input parameters for genetic algorithm

Parameter	Description	Value
population	number of individuals in population	200
generation	number of generations	500
crossover	crossover index (0.1–1.0)	0.9
mutation	mutation index (0.1–1.0)	0.5

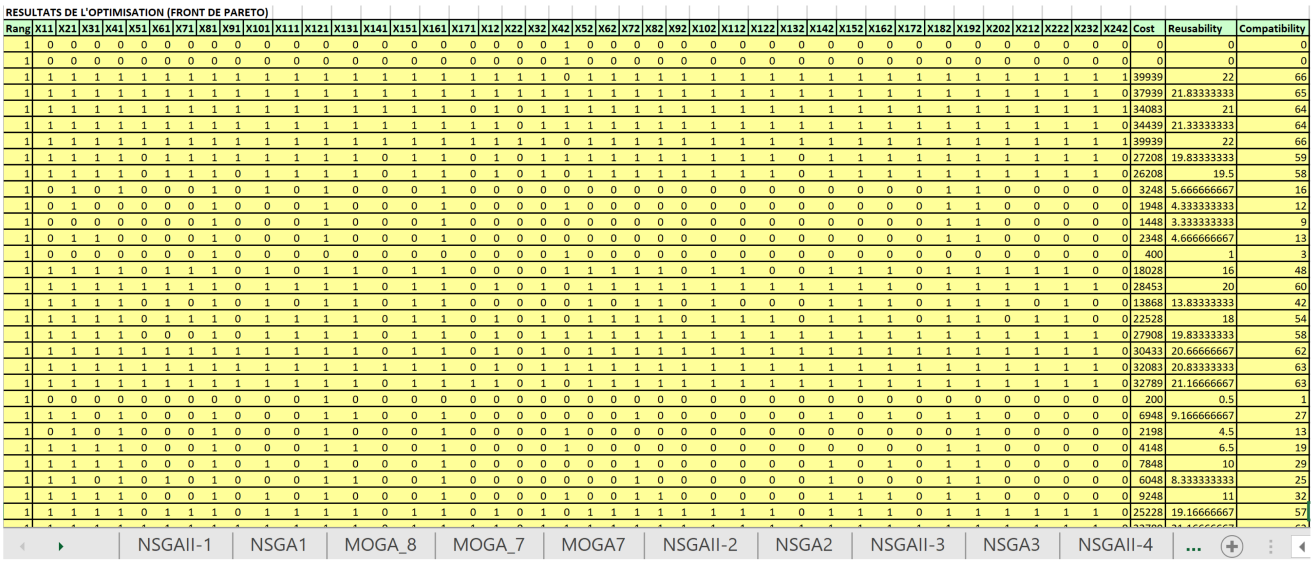

Fig. 6 *Pareto Front*

algorithms. Also, the multi-objective optimisation model offers an alternative to combine features, and it is useful to visualise different valid products. The obtained results prove that the optimisation in software product design gives great benefits not only the reusability but the quality and the final performance of software product, as well as a greater economic benefit by adjusting the production cost to the user's budget.

As future work, to update and introduce new functionalities for MPL is proposed. In the same way, it is convenient to refine and complement the proposed multi-objective optimisation model with new variables in order to customise other software products and covering different market segments.

8 Acknowledgments

The authors would like to acknowledge the support provided by the CONACyT to realise the current research.

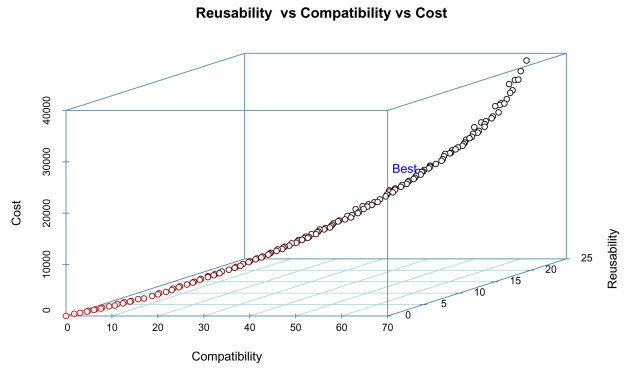

Fig. 7 *Pareto Front obtained by NSGA II algorithm*

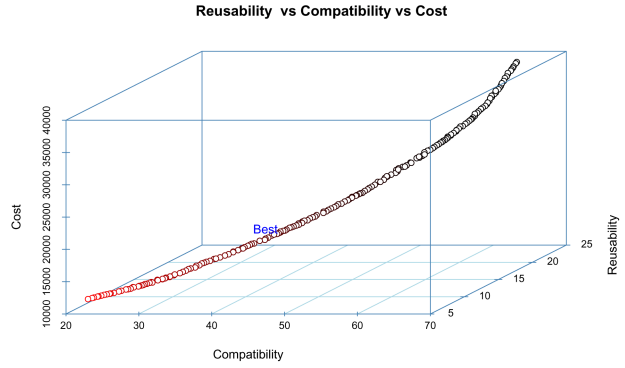

Fig. 8 *Pareto Front obtained by the NSGA II SBX Classique algorithm*

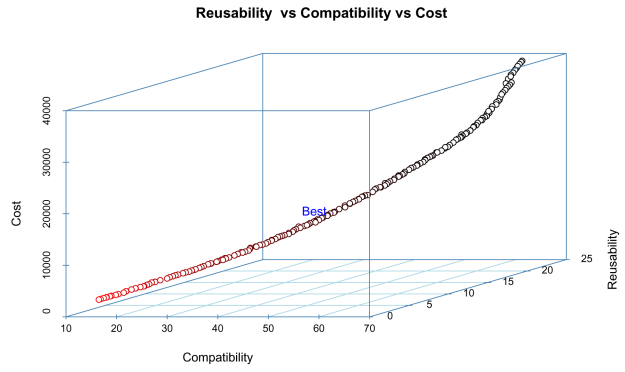

Fig. 9 *Pareto Front obtained by the NSGA II b algorithm*

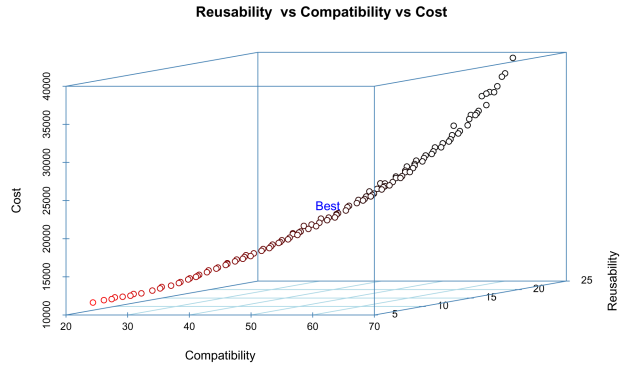

Fig. 10 *Pareto Front obtained by the NSGA II Mixte Continu-Entier algorithm*

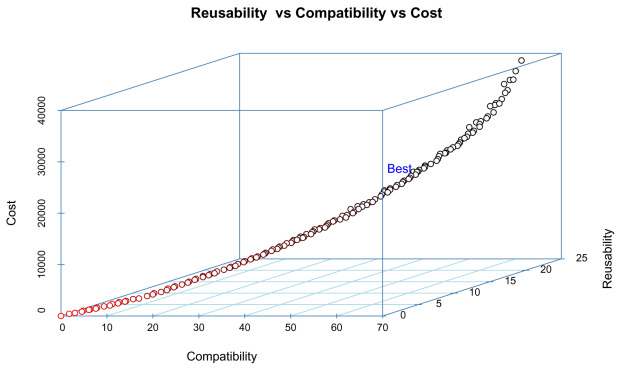

Fig. 11 *Pareto Front obtained by the NSGA II Mixte Continu-Entier-Booléen algorithm*

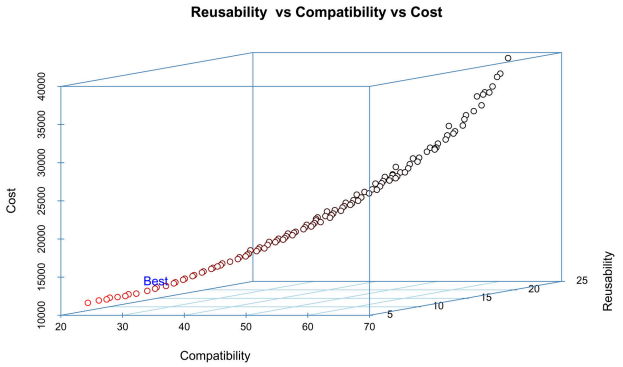

Fig. 12 *Pareto Front obtained by the NSGA II Mixte Continu-Entier-Booléen LBCE algorithm*

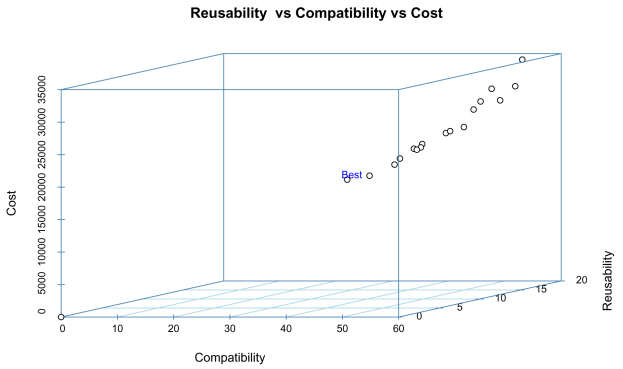

Fig. 13 *Pareto Front obtained by the MIB MOGA Continu-Entier-Booléen (LBCE) algorithm*

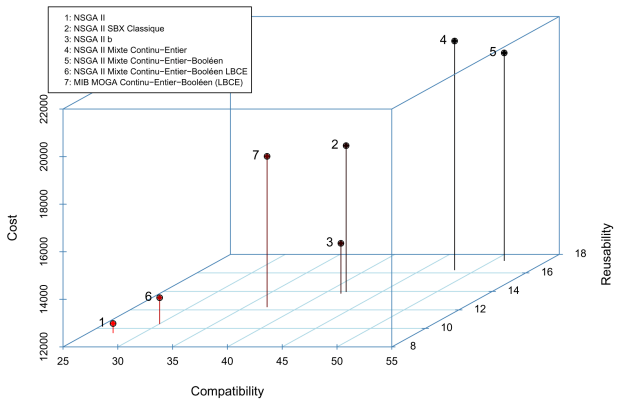

Fig. 14 *Results of solving the optimisation model using GA*

Table 6 Comparative study of the results obtained with genetic algorithms

Algorithm	Cost	Reusability	Compatibility	Run-time, s
NSGA II	12,390.67	9.5276	27.2361	1073
NSGA II SBX Classique	18,135.54	13.9791	41.6974	1380
NSGA II b	14,101.18	13.7945	41.5040	1416
NSGA II Mixte Continu-Entier	21,618	16.3333	48	1353
NSGA II Mixte Continu-Entier-Booléen	20,728	17.3333	51	1060
NSGA II Mixte Continu-Entier-Booléen LBCE	13,098	10.5	30	1016
MIB MOGA Continu-Entier-Booléen	18,325	12.3333	37	2137

Table 7 Optimal configuration by each algorithm

NSGA II	NSGA II SBX Classique	NSGA II b	NSGA II Mixte Continu-Entier	NSGA II Mixte Continu-Entier-Booléen	NSGA II Mixte Continu-Entier-Booléen LBCE	MIB MOGA Continu-Entier-Booléen
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	0
1	1	1	1	1	1	0
0	0	0	0	0	0	1
0	0	0	0	1	0	1
0	0	0	0	0	0	0
1	1	1	1	1	1	1
0	0	0	0	0	0	1
1	1	1	1	1	0	1
0	0	0	0	0	0	0
1	1	1	1	1	1	1
1	1	1	1	1	0	1
0	0	0	0	0	0	0
1	1	1	1	1	0	1
1	1	1	1	1	1	1
0	0	0	0	0	0	1
0	0	0	1	1	1	1
0	0	0	1	0	1	1
0	0	0	1	0	1	1
0	0	0	1	0	1	1
1	1	1	1	1	0	1
0	0	0	0	1	0	1
1	1	1	1	1	1	1
1	1	1	1	1	0	1
0	0	0	0	0	0	1
1	1	1	1	1	0	0
0	0	0	0	1	0	0
0	0	0	0	1	0	1
0	0	0	0	0	0	0
1	1	1	1	1	1	1
1	1	1	1	1	0	1
1	1	1	1	1	0	1
0	0	0	0	0	0	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
0	0	0	1	1	0	1
1	1	1	1	1	0	0
1	1	1	1	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	1

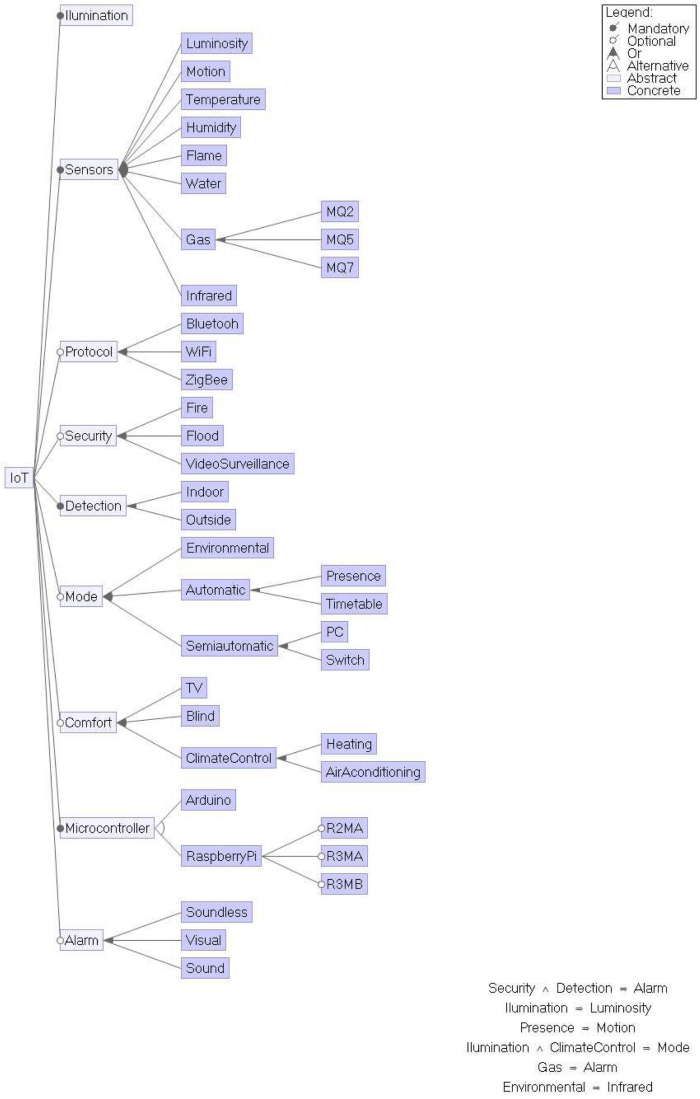

Fig. 15 *MPL-feature model for IoT applications*

9 References

[1] Guo, J., White, J., Wang, G., *et al.*: 'A genetic algorithm for optimized feature selection with resource constraints in software product lines', *J. Syst. Softw.*, 2011, **84**, (12), pp. 2208–2221

[2] Wang, Y.L., Pang, J.W.: 'Ant colony optimization for feature selection in software product lines', *J. Shanghai Jiaotong Univ.*, 2014, **19**, (1), pp. 50–58

[3] Tan, T.H., Xue, Y., Chen, M., *et al.*: 'Optimizing selection of competing features via feedback-directed evolutionary algorithms'. Proc. of the 2015 Int. Symp. on Software Testing and Analysis, New York, NY, USA, 2015, pp. 246–256

[4] Xue, Y., Zhong, J., Tan, T.H., *et al.*: 'IBED: combining ibea and de for optimal feature selection in software product line engineering', *Appl. Soft Comput.*, 2016, **49**, pp. 1215–1231

[5] dos Santos Neto, P.D.A., Britto, R., Rabêlo, R.D.A.L., *et al.*: 'A hybrid approach to suggest software product line portfolios', *Appl. Soft Comput.*, 2016, **49**, pp. 1243–1255

[6] Asadi, M., Soltani, S., Gasevic, D., *et al.*: 'Toward automated feature model configuration with optimizing non-functional requirements', *Inf. Softw. Technol.*, 2014, **56**, (9), pp. 1144–1165

[7] Perrouin, G., Oster, S., Sen, S., *et al.*: 'Pairwise testing for software product lines: comparison of two approaches', *Softw. Qual. J.*, 2012, **20**, (3–4), pp. 605–643

[8] Henard, C., Papadakis, M., Perrouin, G., *et al.*: 'Multi-objective test generation for software product lines'. Proc. of the 17th Int. Software Product Line Conf. on – SPLC '13, New York, New York, USA, 2013, p. 62

[9] Chohan, A.Z., Bibi, A., Hafeez Motla, Y.: 'Optimized software product line architecture and feature modeling in improvement of SPL'. Proc. – 2017 Int. Conf. on Frontiers of Information Technology, FIT 2017, Islamabad, Pakistan, 2018, pp. 167–172

[10] Féderle, E.L., do Nascimento-Ferreira, T., Colanzi, T.E., *et al.*: 'Opla-tool: a support tool for search-based product line architecture design'. Proc. of the 19th Int. Conf. on Software Product Line, SPLC '15, Nashville, TN, USA, 2015

[11] Trujillo-Tzanahua, G.I., Juárez-Martínez, U., Aguilar-Lasserre, A.A., *et al.*: 'Multiple software product lines: applications and challenges', in '*Advances in intelligent systems and computing*', vol. **688**, (Springer, Cham, Switzerland, 2018), pp. 117–126

[12] Khedri, N., Khosravi, R.: 'Towards managing data variability in multi product lines'. Proc. of the 3rd Int. Conf. on Model-Driven Engineering and Software Development, Portugal, 2015, pp. 523–530

[13] Abbas, A., Farah Siddiqui, I., Lee, S.U.-J., *et al.*: 'Multi-objective optimum solutions for IoT-based feature models of software product line', *IEEE. Access.*, 2018, **6**, pp. 12228–12239

[14] Ayala, I., Amor, M., Fuentes, L., *et al.*: 'A software product line process to develop agents for the iot', *Sensors*, 2015, **15**, (7), pp. 15640–15660

[15] Clements, P.C., Northrop, L.: '*Software product lines: practices and patterns*' (Addison-Wesley, Boston, MA, USA, 2001)

[16] Pohl, K., Böckle, G., van der Linden, F.J.: '*Software product line engineering: foundations, principles and techniques*', vol. **10**, (Springer, Berlin, Germany, 2005)

[17] Svahnberg, M., van Gurp, J., Bosch, J.: 'A taxonomy of variability realization techniques', *Softw. - Pract. Exp.*, 2005, **35**, (8), pp. 705–754

[18] Kang, K., Cohen, S., Hess, J., *et al.*: '*Feature-oriented domain analysis (foda) feasibility study*' (Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990), CMU/SEI-90-TR-021

[19] Aoyama, M.: 'Continuous and discontinuous software evolution: aspects of software evolution across multiple product lines'. Proc. of the 4th Int. Workshop on Principles of Software Evolution, New York, NY, USA, 2001, pp. 87–90

[20] Aoyama, M., Watanabe, K., Nishio, Y., *et al.*: 'Embracing requirements variety for e-governments based on multiple product-lines frameworks'. Proc. 11th IEEE Int. Requirements Engineering Conf., Monterey Bay, CA, USA, 2003, p. 285

[21] Bühne, S., Lauenroth, K., Pohl, K.: 'Why is it not sufficient to model requirements variability with feature models?'. Proc. of the automotive requirements engineering AURE04, Nagoya, Japan, 2004

[22] Acher, M., Collet, P., Lahire, P., *et al.*: '*Managing multiple software product lines using merging techniques*' (University of Nice Sophia Antipolis, France, 2010)

[23] Acher, M., Collet, P., Gaignard, A., *et al.*: 'Composing multiple variability artifacts to assemble coherent workflows', *Softw. Qual. J.*, 2012, **20**, (3), pp. 689–734

[24] Holl, G., Grünbacher, P., Rabiser, R.: 'A systematic review and an expert survey on capabilities supporting multi product lines', *Inf. Softw. Technol.*, 2012, **54**, (8), pp. 828–852

[25] Hartmann, H., Trew, T.: 'Using feature diagrams with context variability to model multiple product lines for software supply chains'. Proc. of the 12th Int. Software Product Lines Conf., Washington, DC, USA, 2008, pp. 12–21

[26] Tan, L., Lin, Y., Ye, H.*, et al.*: 'Improving product configuration in software product line engineering'. Conf. in Research and Practice in Information Technology Series, Adelaide, Australia, 2013, vol. **135**, pp. 125–134

[27] Cruz, J., Neto, P.S., Britto, R.*, et al.*: 'Toward a hybrid approach to generate software product line portfolios'. 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, 2013, pp. 2229–2236

[28] Afzal, U., Mahmood, T., Rauf, I.*, et al.*: 'Minimizing feature model inconsistencies in software product line'. IEEE INMIC 2014 – Proc. 17th IEEE Int. Multi Topic Conf.: Collaborative and Sustainable Development of Technologies, Karachi, Pakistan, 2014, pp. 137–142

[29] Zhan, Z., Luo, W., Guo, Z.*, et al.*: 'Feature selection optimization based on atomic set and genetic algorithm in software product line', in '*Advances in intelligent systems and computing*', vol. **686**, (Springer, Cham, Switzerland, 2018), pp. 93–100

[30] Font, J., Arcega, L., Cetina, Ø.H.: 'Feature location in model-based software product lines through a genetic algorithm', in '*Software reuse: bridging with social-awareness*' (Springer International Publishing, Cham, Switzerland, 2016), pp. 39–54

[31] Wang, S., Ali, S., Gotlieb, A.: 'Cost-effective test suite minimization in product lines using search techniques', *J. Syst. Softw.*, 2015, **103**, pp. 370–391

[32] Alsariera, Y.A., Majid, M.A., Zamli, K.Z.: 'Splba: an interaction strategy for testing software product lines using the bat-inspired algorithm'. 2015 4th Int. Conf. on Software Engineering and Computer Systems, ICSECS 2015: Virtuous Software Solutions for Big Data, Kuantan, Malaysia, 2015, pp. 148–153

[33] Alsawalqah, H.I., Kang, S., Lee, J.: 'A method to optimize the scope of a software product platform based on end-user features', *J. Syst. Softw.*, 2014, **98**, pp. 79–106

[34] Colanzi, T.E., Vergilio, S.R., Gimenes, I.M.S.*, et al.*: 'A search-based approach for software product line design'. Proc. of the 18th Int. Software Product Line Conf. on - SPLC '14, New York, USA, 2014, pp. 237–241

[35] Gomez, A., Pibouleau, L., Azzaro-Pantel, C.*, et al.*: 'Multiobjective genetic algorithm strategies for electricity production from generation {IV} nuclear technology', *Energy Convers. Manage.*, 2010, **51**, (4), pp. 859–871