Towards a Microservices-Based Product Line with Multi-Objective Evolutionary Algorithms

Willian D.F. Mendonça*, Wesley K.G. Assunção†‡, Lucas V. Estanislau†, Silvia R. Vergilio*, Alessandro Garcia‡
*DInf, Federal University of Paraná (UFPR). Curitiba, Brazil
†COTSI, Federal University of Technology - Paraná (UTFPR). Toledo, Brazil
‡DI, Pontifical Catholic University of Rio de Janeiro (PUC-Rio). Rio de Janeiro, Brazil
williandouglasferrari@gmail.com, wesleyk@utfpr.edu.br, estanislau@alunos.utfpr.edu.br, silvia@inf.ufpr.br, afgarcia@inf.puc-rio.br

***Abstract*—Microservices are small and independently deployable services. They can be developed on different platforms and communicate via lightweight protocols, what makes them highly interoperable. The interoperability between microservices, as well as their reuse and customization needs make this kind of systems adequate to constitute a Software Product Line. However, there is no automatic approach to support the designing of Microservices-Based Product Lines (MBPLs). To move towards the development of MBPLs, this work presents an approach, named MOEA4MBPL, to extract Feature Models (FMs) from a set of microservices-based systems. These FMs intent to leverage interoperability, enabling the practitioners to reason about reuse and/or customization of functionalities. The proposed approach is based on multi-objective evolutionary algorithms, optimizing three objectives, namely precision and recall of products denoted by an FM, and conformance with existing dependencies between microservices. MOEA4MBPL was evaluated with six microservices-based systems, using the algorithms NSGA-II and SPEA2. Our approach was capable of finding FMs with good trade-off values of precision and recall, satisfying all dependencies among the microservices. SPEA2 found better fronts of solutions than NSGA-II, but the latter always executed faster and could find single solutions closer to an ideal solution than the former.**

***Index Terms*—microservices, multi-objective optimization, variability model, product line.**

I. Introduction

Microservices are a software development technique that has been adopted by many companies in recent years [1]. Microservices are small, autonomous and independently deployable services that work together [2]. The benefits of microservices-based systems are [3]: reduced effort for maintenance and evolution, increased availability of services, ease of innovation, continuous delivery, ease of DevOps incorporation, and facilitated scalability.

Another advantage of microservices-based systems is the heterogeneous interoperability, which refers to the ability to integrate parts of different systems implemented in different programming languages and platforms [4]. For example, processing a complex business rule, which requires the coordination among a Java-based application, a PHP application, and a COBOL program, requires a complex solution in a monolithic architecture, but would be easier managed in a microservices-based architecture. This is a common situation in many companies that deal with different platforms and legacy systems. Interoperability has also an appealing characteristic of providing easier reuse of functionalities. A recent study with practitioners has pointed that microservices also have the goal of supporting reuse and customization [1]. Such a study stated that variability is a key criterion for structuring a microservices-based architecture.

Reuse and customization are the core goal of Software Product Lines (SPLs) [5]. Then, some studies in the literature have taken into account these characteristics to model microservices-based systems as SPLs [6], [7], [8]. However, such studies do not provide an automatic approach to support the designing of a *Microservices-Based Product Line (MBPL)*.

Given the aforementioned limitation, we propose in this paper an approach named *MOEA4MBPL (Multi-Objective Evolutionary Algorithm For Microservices-Based Product Line)*. MOEA4MBPL extracts variability models, a.k.a. Feature Models (FMs), from different microservices-based systems. Such FMs can be used for defining an MBPL, leveraging interoperability and allowing artifacts reuse in different systems. Our approach is based on Multi-Objective Evolutionary Algorithms (MOEAs), and employs an evolutionary process to find a set of FMs with trade-off that best represent the actual microservices-based systems we are dealing with. The goal is to maximize precision (deriving only desired systems), recall (including all desired systems), and conformance (with respect to existing dependencies between microservices).

MOEA4MBPL was evaluated with six microservices-based systems, using the MOEAs *Non-Dominated Sorting Genetic Algorithm (NSGA-II)* [9] and *Strength Pareto Evolutionary Algorithm (SPEA2)* [10]. Our approach was capable of finding FMs with good trade-off values of precision and recall, satisfying 100% of the dependencies among the microservices. SPEA2 found better fronts of solutions than NSGA-II, but the latter always executed faster and could find single solutions closer to an ideal solution than the former.

The contribution of this paper is to leverage the benefits of microservices-based systems, mainly related to interoperability. We want to benefit from the fact that the microservices-based architecture style attempts to reduce the number of choices for functionalities integration. Our approach is a first step to enable reuse of microservices in different systems to enhance the functionalities available in a software system. The FMs can provide a general view of microservices organization among different systems, enabling practitioners to reason about reuse and/or customization of functionalities.

This paper is structured as follows. Section II presents related work. Section III describes MOEA4MBPL. Section IV presents the setup followed to evaluate our approach. Results are presented and analyzed in Section V. Finally, Section VI concludes the paper and outlines future work.

II. Related Work

Related work can be classified in two main categories: (i) studies from the SPL field and (ii) pieces of works from the Service-Oriented Product Line (SOPL) field [11].

In the first category, we find many studies to reverse engineer FMs from feature sets [12], [13], [14], [15], [16], [17], [18]. The most promising ones apply multi-objective optimization. Despite being different problems, the main ideas of these works can be explored in the context of microservices. Considering that, our work is inspired by the approach of Assunção et al. [17], [18], which uses multi-objective evolutionary algorithms to extract variability-safe FMs based on code dependencies of system variants. But differently from these similar works in the SPL context, we address another problem, which may have impact in the behavior of algorithms.

Some authors [19] state the idea that microservices arose by re-conceiving service-oriented computing. Then, there is a relation between SOPL and the idea of MBPL. Lee et al. [20] propose an approach that uses FM analysis to identify services, but such an approach does not apply search-based techniques. The work of Khosnevis and Shamn [21] receives as input a graph of business activities (BPFM - Business Process Family Model) and uses the NSGA-II algorithm to identify the services and variabilities, considering metrics such as cohesion, coupling, granularity, business entity convergence, and commonality degree. Differently, our work extracts FMs from functionalities of existing microservices-based systems.

We can find studies that integrate the use of microservices-based architectures and SPL approach to develop, for instance, a multi-tenant SasS (Software as a Service) [8] and WISE (Weather In Sights Environment) system [7]. The work of Naily et al. [6] proposes a framework for microservices-based software taking into account the Software Product Line Engineering (SPLE). The framework process includes a domain analysis step where the FM is derived. However, the mentioned studies do not provide a general automatic approach.

Analyzing the categories of related work, we can see that our approach, described in the next section, is the first initiative to extract FMs from existing services towards an MBPL.

III. Extracting Feature Models for MBPLs

MOEA4MBPL uses multi-objective optimization to extract a set of FMs that describe variability among microservices-based systems, having as basis previous work in the SPL context [17], [18]. Instead of dealing with variants of a single system and their features, in the work herein described we have as input entire different systems, from the same domain, and their microservices. By leveraging the microservices characteristics of interoperability we move towards enhance the microservices-based systems with new services/functionalities. Details of the proposed approach are presented next.

A. Input

MOEA2BPL uses two input artifacts:

- **Matrix of systems and microservices** where the systems are in the rows and all existing microservices are in the columns. In the cells we indicate which microservice belongs to each system. An example of this matrix, used in the evaluation of the approach, is presented in Table II.
- **Graph of dependencies between microservices** where dependencies existing between microservices are represented. Dependencies are identified based on existing service requests. An example of this graph, used in the evaluation of the approach, is presented in Figure 1.

B. Output

To design an SPL we use an FM to organize the microservices according to their dependencies and variabilities. However, usually there is no a single solution for the problem of defining an FM to represent a set of existing systems. This is the reason for using a multi-objective approach.

MOEA4MBPL produces as output a set of FMs with different trade-offs among three objectives, described below.

C. Objective functions

In our approach the MOEAs optimize three objective functions. To compute them we defined two auxiliary functions. Let us consider $\mathcal{FM}$ as the universe of feature models, $\mathcal{MSS}$ the universe of microservices-based systems, and mss a set of microservices-based systems informed as input. Based on this terminology, we introduce the function *msSystems*:

Definition 1: $msSystems(fm)$ returns the set of microservices-based systems (i.e. products) denoted by a feature model fm.

To check the conformance of the FMs with dependencies between microservices, we define the function *holds*:

Definition 2: $holds(dep, ms)$ returns 1 if dependency dep holds on the microservice-based system ms, and 0 otherwise. A dependency dep holds for a microservice-based system ms if the microservices involved in dep are also in conformance with the tree relationship in the FM.

Considering these two auxiliary functions, in the following we describe the objective functions of our approach.

Definition 3: **Precision (P)** returns how many of the microservices-based systems denoted by a feature model fm are among the desired microservices-based systems mss.

$$precision(mss, fm) = \frac{|mss \cap msSystems(fm)|}{|msSystems(fm)|}$$

Definition 4: **Recall (R)** returns how many of the desired microservices-based systems mss are denoted by a feature model fm.

$$recall(mss, fm) = \frac{|mss \cap featureSets(fm)|}{|mss|}$$

Definition 5: **Dependency Conformance (DC)** expresses the degree of conformance of a feature model fm with a dependency graph dg.

$$dependencyConformance(fm, dg) = \sum_{dep \in dg} dep \times \left(\frac{\sum_{fs \in featureSets(fm)} holds(dep, fs)}{|featureSets(fm)|} \right)$$

All the three objective functions were designed for a maximization problem, where the values are between [0,1].

D. Representation of Individuals and Initial Population

To represent the individual we use a simplified version of the SPLX meta-model[1], which defines both structure and semantic of FMs (similar to [18]). For this, a set of composite objects inherited from a generic class `Feature` describes the tree-like structure of the FM. Another set of objects were designed to represent the *Cross-Tree Constraints (CTCs)* between the features[2]. Based on this representation, the initial population is generated by creating random feature trees and random CTCs. For this task we used the tools FaMa [22] and BeTTy [23].

E. Genetic Operator

To select individuals for mutation and crossover, we employ standard tournament selection. In the evolutionary process there are some domain constraints to ensure the semantics of the solutions. These constraints are: (i) every feature must appear exactly once in the FM tree; (ii) all FMs have a fixed set of features, so in different FMs only the relations between features are different; (iii) CTCs can only be either `requires` or `excludes`; (iv) CTCs must not contradict each other; (v) there is a maximum number of CTCs that must not be exceeded. These domain constraints were not designed to consider cases of contradictions between CTCs and FM tree. In such cases, the evolutionary process will discard the solutions because of their bad fitness value.

1) Mutation: The individuals are mutated by applying modifications in randomly selected parts of the FM tree or in the CTCs (similar to [18]). The modification is randomly selected from the following lists:

- Mutations performed on the tree:
  - Swap two features in the feature tree;
  - Change an *Alternative* group to an *Or* or vice-versa;
  - Change an *Optional* or *Mandatory* relation to any other kind of relation (*Mandatory*, *Optional*, *Alternative*, *Or*);
  - Move a sub-tree in the FM tree to somewhere else without violating the meta-model or any of the domain constraints.
- Mutations performed on the CTCs:
  - Add a new, randomly created CTC;
  - Randomly remove a CTC.

[1]http://www.splot-research.org/

[2]Features are the building blocks for Software Product Lines, which in our context are microservices.

2) Crossover: The crossover generates offspring in conformance to the meta-model representation and to the domain constraints (similar to [18]). The crossover process is:

1) Initialize the child with the root feature of $Parent_1$. If the root feature of $Parent_2$ is a different one then it is added to the child as a mandatory feature of its root feature;
2) Traverse the first parent depth first starting at the *root* and add to the child a random number of features that are not already contained by appending them to their respective parent feature already contained in the child using the same relation type between them;
3) Traverse the second parent similarly as the first one;
4) Go to Step 2 until every feature is contained in the child.

The second child is obtained by performing the same process but with reverse parents. The crossover of CTCs is performed by merging all the constraints of both parents and then randomly assigning a subset of to the first child and the remaining to the second child.

IV. Evaluation Setup

This section describes how MOEA4MBPL was evaluated. For further studies, we make available[3] a package with the raw data results, scripts, and tools herein described.

A. Microservices-based Systems

Our evaluation relies on six open source microservices-based systems. These systems are all from the same domain, namely virtual stores, and were developed by different developers. The subject systems are: Hipster Shop[4], Sock Shop[5], eShopOnContainers[6], Vert.x Micro-shop[7], Shopping Cart[8], and Stan's Robot Shop[9].

To collect the input data for our approach, we identified the characteristics of each microservices-based system. First, we downloaded all projects, then started checking if there was any kind of architecture, that is, a documented architecture. We found some basic images with some microservices names, calling instructions, and infrastructure components. Not all systems have this architecture, then the source code was used for all systems, which are developed in different programming languages such as GO, JavaScript, Python, and Java. In addition to the source code, we read the available documentation, also used in the collection of the input data for MOEA4MBPL.

B. Identified Microservices and Dependencies

Following the methodology presented in the previous section, we identified 14 different microservices/features, which are incorporated in different configurations on the six subject systems. We named these features with a brief description of each one in Table I.

[3]https://wesleyklewerton.github.io/CEC2020_Evaluation_Package.zip
[4]https://github.com/GoogleCloudPlatform/microservices-demo
[5]https://github.com/microservices-demo/microservices-demo
[6]https://github.com/dotnet-architecture/eShopOnContainers
[7]https://github.com/sczyh30/vertx-blueprint-microservice
[8]https://github.com/thangchung/ShoppingCartDemo
[9]https://github.com/instana/robot-shop

TABLE I
MICROSERVICES IDENTIFIED FOR ALL SIX SYSTEMS.

Name	Description
Frontend	Microservice used as user interface and responsible for calling other microservices.
User	Manages the customers and user authentication.
Payment	Manages credit card information and payment tasks with the specified amount and usually returns a transaction ID.
Catalog	Provides the list of products and the ability to search for products.
Cart	Stores items in the user's shopping cart.
Order	Retrieves user cart, prepares order and requests payment, shipment and email notification.
Product	Provides detailed information about products.
EmailContact	Sends email to customers/users.
Shipping	Provides shopping cart-based shipping cost estimates.
Marketing	Shows advertisements based on certain context words.
Recommendation	Recommends other products based on what is provided in the cart.
Audit	Allows administrators to audit user data based on property database and external servers.
Currency	Converts a cash amount into another currency.
Location	Retrieves user location to support advertisements and currency conversion.

The matrix with the six subject systems and their features is presented in Table II. The features `Frontend`, `Payment`, `Catalog` and `Cart` are mandatory in all systems. On the other hand, the other features vary among the systems.

To identify dependencies between microservices and features, we analyzed the information available in the projects, as aforementioned. However, not all systems have clear dependency information. Some systems have a service recorder in their model, and there is no direct and explicit call to a particular microservice. For instance, we had to deeply analyze the source code for eShopOnContainers and Stan's Robot Shop to collect more accurate information.

We could observe that the number of dependencies is quite varied. This information was revealed based on the call directions between microservices found in the basic documented architecture and in the deep analysis of the source code. Figure 1 presents the dependency graph obtained for all systems. There are many dependencies between the microservices, in some cases in both directions. This is a complex situation to serve as benchmark to evaluate the ability of our approach to find FMs in conformance with these dependencies.

The graph analysis reveals that `Frontend` and `Order` require several other microservices for their functionality. `Order` has eight dependencies and three microservices depend on it, whereas `Frontend` depends on ten microservices and five microservices depend on it.

C. MOEAs and Parameters

We applied NSGA-II and SPEA2 in our evaluation. NSGA-II adopts an elitism strategy classifying the solutions according

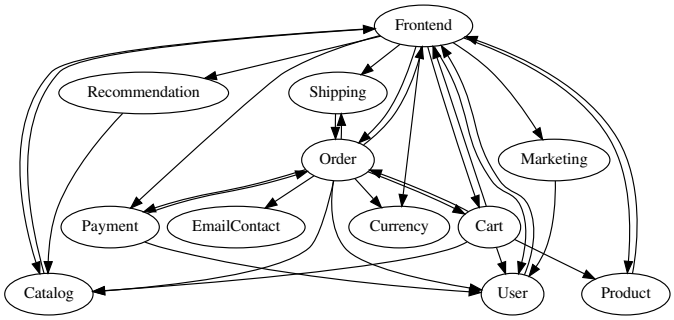


Fig. 1. Dependencies identified considering call requests between microservices for all six systems used in the evaluation.

their dominance. SPEA2 uses an external archive to create the fronts of non-dominated solutions. These algorithms are commonly applied in search-based software engineering approaches [24]. For the experimentation we rely on the implementations available on ECJ Framework[10].

Table III presents the parameter settings for both algorithms. The number of generations and the archive size parameters were empirically calibrated. The remaining parameters are the same adopted in [18]. We performed 30 independent runs for each algorithm on a machine with an Intel® Core^{TM} i7-3632QM CPU with 2.2 GHz, 16 GB of memory, and running on a Linux platform.

D. Quality Indicators and Statistical testing

We used the well-known quality indicator Hypervolume (HV) to reason about the differences between NSGA-II and SPEA2 [25]. HV was computed by using the recursive and dimension-sweep algorithm implementation[11] [26]. The reference point to compute HV was P=1.1, R=1.1, and DC=1.1.

We used the indicator Euclidean Distance from Ideal Solution (ED) to find the closest solutions to the best theoretical objectives, i.e. an ideal solution [27]. For our maximization problem, an ideal solution has the maximum value of each objective, that is, P=1.0, R=1.0, and DC=1.0.

In the comparison of different computational intelligence algorithms, the number of fitness evaluations is usually the basis to measure computational effort to reach solutions. In addition, we also considered runtime to analyze how fast/slow the solutions were reached. We collected the runtime, in milliseconds, of each independent run for each MOEA.

To check statistical difference between NSGA-II and SPEA2 we applied the Wilcoxon signed-rank test [28]. To corroborate our analysis we also compute the effect size with the Vargha-Delaney's $\hat{A}_{12}$ measure [29]. Both Wilcoxon and $\hat{A}_{12}$ are commonly used for assessing randomized algorithms in Software Engineering [24], [30].

[10]http://cs.gmu.edu/~eclab/projects/ecj/
[11]http://lopez-ibanez.eu/hypervolume

TABLE II
MICROSERVICES IDENTIFIED FOR EACH ONE OF THE SIX SYSTEMS USED IN THE EVALUATION.

Microservices	Systems					
	Hipster Shop	**Sock Shop**	**eShopOn-Containers**	**Vert.x Micro-shop**	**Shopping Cart**	**Stan's Robot Shop**
Frontend	✓	✓	✓	✓	✓	✓
User		✓	✓	✓	✓	✓
Payment	✓	✓	✓	✓	✓	✓
Catalog	✓	✓	✓	✓	✓	✓
Cart	✓	✓	✓	✓	✓	✓
Order	✓	✓	✓	✓	✓	
Product				✓		
EmailContact	✓	✓				
Shipping	✓	✓				✓
Marketing	✓		✓			
Recommendation	✓					
Audit					✓	
Currency	✓					
Location			✓			

TABLE III
PARAMETER CONFIGURATIONS USED TO EXECUTE THE ALGORITHMS NSGA-II AND SPEA2 DURING THE EVALUATION.

Parameters	**NSGA-II**	**SPEA2**
Number of Generations	1000	1000
Population Size	500	500
Archive Size	-	50
Crossover Rate	0.7	0.7
Tree Mutation Rate	0.5	0.5
CTCs Mutation Rate	0.5	0.5
Number of Elites	25%	25%
Tournament Size	6	6
Maximum CTC Percentage for Builder*	0.1	0.1
Maximum CTC Percentage for Mutator*	0.5	0.5
Independent runs	30	30

* relative to number of features

TABLE IV
RESULTS OF HYPERVOLUME (HV), EUCLIDEAN DISTANCE FROM IDEAL SOLUTION (ED), AND RUNTIME OF THE 30 INDEPENDENT RUNS.

Indicator	**Average (Std dev.)**		**Wilcoxon**	**$\hat{A}_{12}$ Effect Size**	
	NSGA-II	**SPEA2**	**p-value**	**NSGA-II**	**SPEA2**
HV	0.0275 (0.0112)	0.0123 (0.0084)	1.01E-04	21.00%	79.00%
ED	0.5051 (0.1332)	0.6864 (0.1371)	1.36E-05	82.39%	17.61%
Runtime	46.363,00 (5.701,84)	482.707,20 (126.951,86)	1.69E-17	100.00%	0.00%

V. RESULTS AND ANALYSIS

This section presents the results and analysis based on quality indicators mentioned in the last section. In addition, we present analysis of the best solutions found by each MOEA.

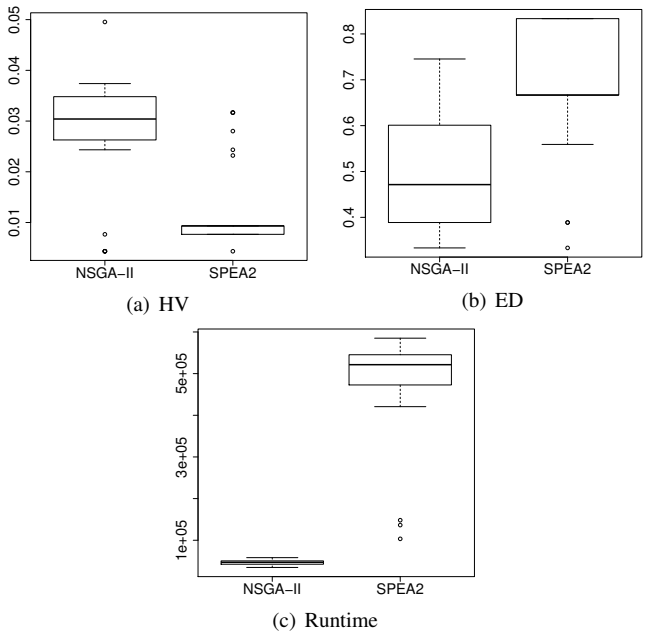
Table IV presents the results of HV, ED, and Runtime considering the 30 independent runs or each MOEA. The first two columns show the average and standard deviation, in parentheses. Since we are dealing with a maximization problem, lower values of HV are better. Runtime is computed in milliseconds. The p-value and effect size of the comparison between the 30 values of HV for each MOEA are shown in the last three columns. For a deeper comparison, Figure 2 also presents the boxplots for these three indicators.

From the results of Table IV we observe that SPEA2 reached an average value of HV lower than the half of the NSGA-II HV value. For corroborating our analysis, in Figure 2(a) we can see that removing the outliers the difference is great, being the values of HV reached by SPEA2 lower than one-third when compared to NSGA-II. The p-value, that is lower than 0.05, indicated significant difference with confidence of 95%. The $\hat{A}_{12}$ measure pointed a difference of large magnitude, where there is 79% of chances SPEA2 reaches better results than NSGA-II. For the indicator ED the difference on average was not so great, see Figure 2(b), but still there is statistical difference, as pointed by Wilcoxon test and effect size measure. Regarding ED, NSGA-II is the best MOEA, reaching better solutions than SPEA2 in more than 82% of the runs. When considering the single solution with the best trade-off among the fitness objectives, NSGA-II is also better.

As mentioned in Section IV, we configured both MOEAs with the same number of fitness evaluations, namely 500.000 (1000 generations * 500 individual). In addition, we also collected the runtime for analysis, as shown in the last row of Table IV. Here the algorithm NSGA-II was statistically better than SPEA2, with the effect sizes showing that NSGA-II has a lower runtime in 100% of the cases. This happens mainly because SPEA2 uses a strategy based on archive, which makes its evolutionary process slower.

We performed an analysis related to the number of solutions found by each MOEA, considering the following sets of solutions: (i) PF_{approx}: the solutions found in each independent run; (ii) PF_{known}: has the non-dominated solutions considering the union of all solutions of the 30 runs, i.e. the 30 PF_{approx} sets of each MOEA; and (iii) PF_{true}: contains the non-dominated solutions considering the PF_{known} sets of NSGA-II and SPEA2, i.e. the best solutions found for our problem.

On average NSGA-II found 373.50 (std dev. = 114.46) and SPEA2 321.10 (std dev. = 86.21) solutions per run. We



(a) HV (b) ED

(c) Runtime

Fig. 2. Boxplot of the Hypervolume (HV), Euclidean Distance from Ideal Solution (ED), and Runtime of the 30 independent runs of NSGA-II and SPEA2. For the three indicators, lower values are better.

observed that many of these solutions in the PF_{approx} have similar values of objectives, but different FMs. Considering the PF_{known}, each MOEA found three non-dominated solutions. These solutions are presented in the search-space in Figure 3. Two solutions in the PF_{known} of both MOEAs have the same objective values: (P=1.0, R=0.666, DC=1.0) and (P=0.555, R=0.833, DC=1.0). Taking the additional solutions of each MOEA, the SPEA solution (P=0.352, R=1.0, DC=1.0) dominates the NSGA-II solution (P=0.25, R=1.0, DC=1.0). In summary, the PF_{true} is composed of three solutions, three found by SPEA2, and only two found by NSGA-II. Regarding the best solutions, NSGA-II found one solution belonging to PF_{true} in seven independent runs, on the other hand, SPEA2 found the best solutions in only two, where in one case SPEA2 found one solution and in another case two.

Figure 4 presents some FMs[12], solutions with the best values for the objective functions. Figures 4(a) and 4(b) were found by both MOEAs, Figure 4(c) by SPEA2, which together with the two previous ones form the set PF_{true}. Figure 4(d) was found by NSGA-II, but this solution is dominated by the one of Figure 4(c). To illustrate the characteristics of one solution, we will analyze Figure 4(a). Let us consider we will take this FM for constructing a virtual store MBPL. By the FM denoted in this figure we can derive 4 virtual stores, which all of them are in the set of systems used as input, representing a P=1.0. However, 2 systems would be missing, leading to a R=0.66. All dependencies between microservices are respected in this microservices-tree organization (DC=1.0). On the other hand, with the FM presented in Figure 4(c) we can derive 17 virtual stores, having all the six input systems, which leads to R = 1.0. However, the surplus of 11 systems decreases the value of P, which is equal to 0.35. All solutions presented in Figure 4 have DC=1.0, which indicates they are in conformance with the dependencies identified in the input systems, shown in Figure 1. This is important, since the migration to a MBPL using these FMs will not require to break any already existing dependence among the microservices.

It is important to notice that in the FMs presented to illustrate our approach there are redundant cross-tree constraints, marked with a warning symbol (⚠) and observed in the constraint ¬`Location` ∨ `Frontend` in Figure 4(a). Such situation happens during the evolutionary process, but the FMs can be refined by the user, to remove this incoherence.

The results presented and analyzed in this section show that SPEA2 was better than NSGA-II to find fronts of solutions with the best values of objectives (shown by HV indicator). Interestingly, SPEA2 found the best solutions in only two of the 30 independent runs, but all of them composed the PF_{true}. On the other hand, NSGA-II was absolutely (in 100% of the cases) the fastest algorithm of our evaluation, and found the single solutions (in PF_{approx}) closer to an ideal point (shown by ED indicator). Finally, MOEA4MBPL is capable to find FMs with high values of precision and recall (denoting the desired microservices-based systems), and satisfying 100% of the dependencies between the microservices.

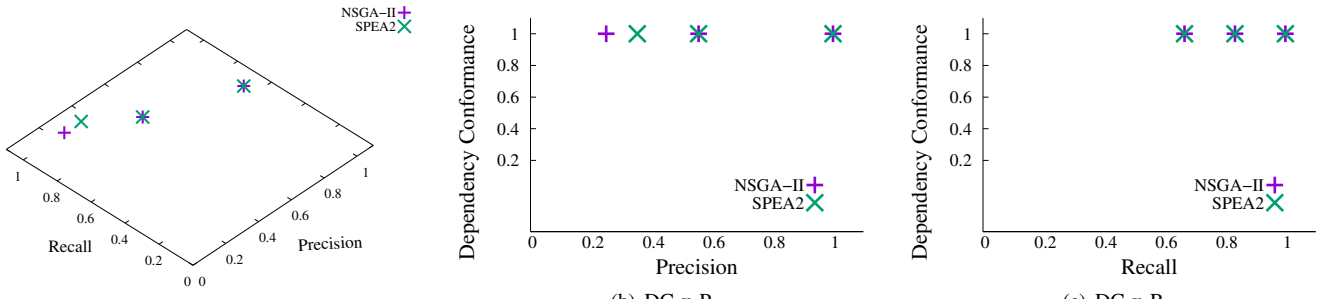
The discussions presented above are related to the optimization process. In a practical point of view, we can discuss the advantages of proposing an SPL based on different microservices-based systems in comparison to SPLs obtained from system variants. When an FM is extracted from system variants, usually the features represent common implementation of building-blocks, with only small modifications, that will be used to configure different products. In an FM obtained with MOEA4MBPL we also can reason about the configuration of different products. In addition to this, since our input are entire different systems, we can experiment different microservices/features implementations. This allows us to select the best implementations for the context we have. For example, feature `Payment` is present in all six web stores, with different implementations. We can try these different implementations and choose the one that best fits our context. All this experimentation of different implementations is supposed to be an easy task due to the interoperability between microservices.

VI. Concluding Remarks

This paper introduced MOEA4MBPL, an approach to extract FMs from a set of microservices-based systems. The approach relies on multi-objective and evolutionary algorithms and generates a set of FMs with good trade-off of three objectives: recall, precision, and conformance with dependencies between the microservices.

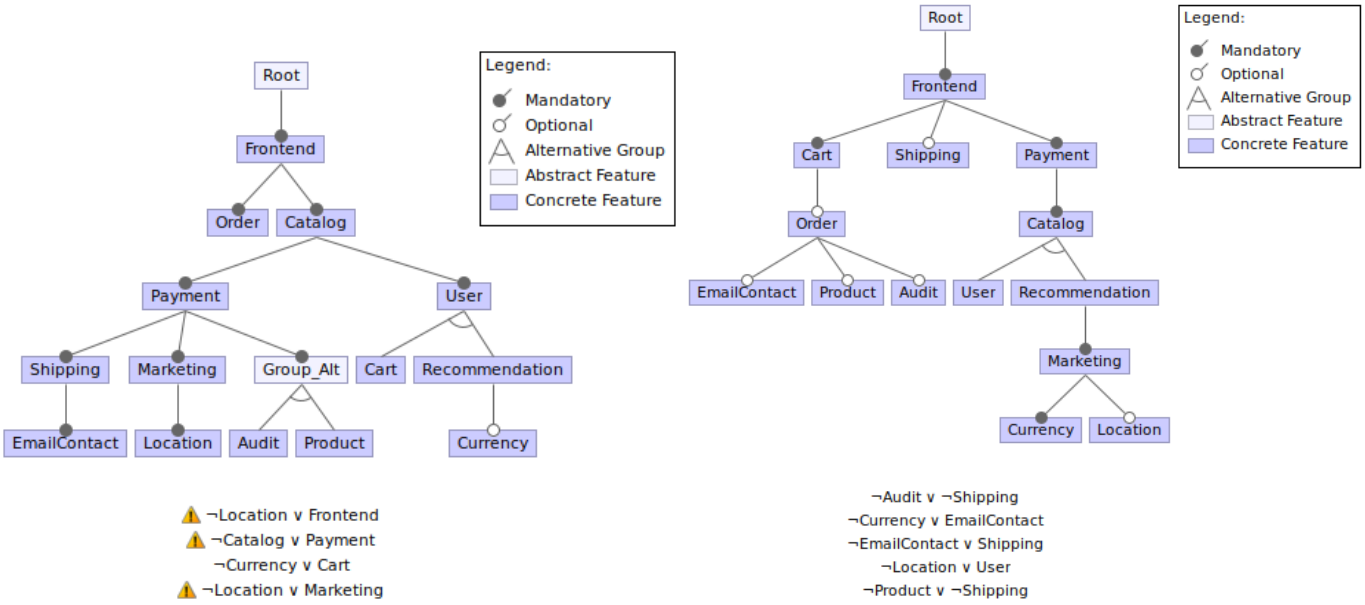
In the evaluation, MOA4MBPL obtained solutions with high objective values. The generated FMs represent all the desired microservices-based systems and do not violate any existing

[12]This FM-tree like representation was created using FeatureIDE: https://featureide.github.io/

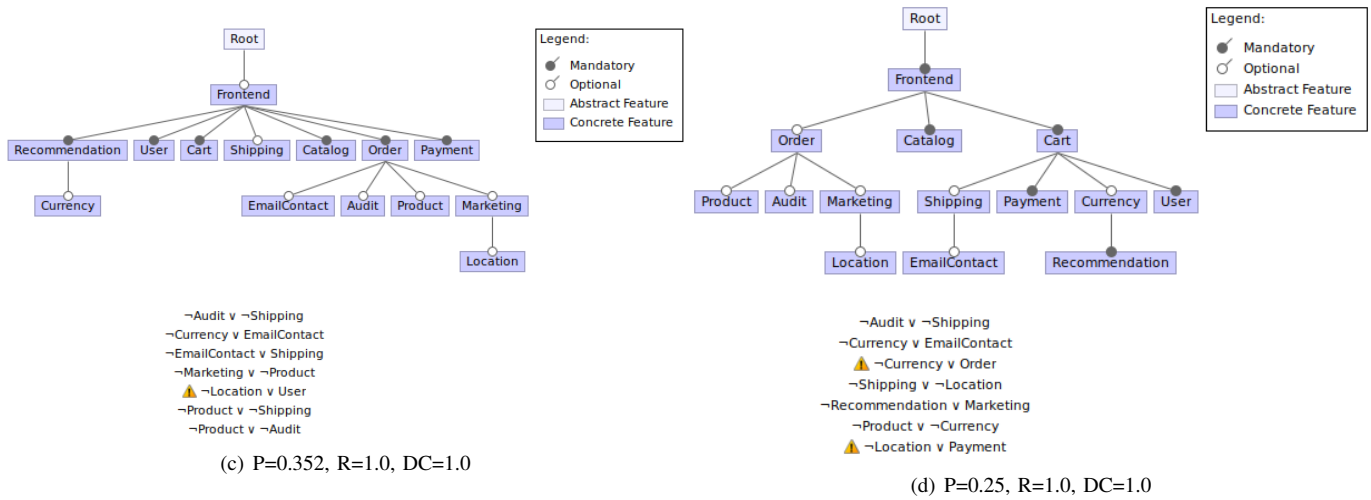


(a) P x R (b) DC x P (c) DC x R

Fig. 3. Three solutions found by NSGA-II and three solutions found by SPEA2 on the Search Space in different combinations of objectives. Two solutions found by both algorithms have the same values for the three objectives, and one solution of SPEA2 has a better value of P, dominating the solution of NSGA-II.



(a) P=1.0, R=0.666, DC=1.0 (b) P=0.555, R=0.833, DC=1.0



(c) P=0.352, R=1.0, DC=1.0 (d) P=0.25, R=1.0, DC=1.0

Fig. 4. Feature Models of the best solutions found by NSGA-II and SPEA2. Solutions (a) and (b) were found by both algorithms. Solution (c) was found only by SPEA2 and dominates the solution (d) found by NSGA-II.

dependency. From this set, the software engineers can select the best solution according to their needs. SPEA2 presented the best value regarding HV and more solutions in PF_{true}. On the other hand, NSGA-II presented results with the best value of ED and runtime. The approach reached good results with both algorithms. However, deeper analysis should be conducted in the future, including other MOEAs and systems.

We had some insights during the conduction of this study. SPLs are commonly implemented using compositional or annotative approaches [31]. In the context of microservices, we have a third option, which is the use of APIs to integrate implementation artifacts. Microservices seem to be adequate to implement SPLs because of it well-modularization and interoperability. However, the variability management in a multi-tenancy environment is a complex situation that needs to be deeper investigated.

Future work should also consider other objective functions and other kind of information regarding the microservices such as non-functional properties, as well as the creation of other architectural models required for an MBPL. For example, dependency conformance deals with dependencies between microservices, but it could be based on other types of implementation artifacts, or be given by domain experts based on their knowledge of the microservices-based systems. Our work represents just a first step in this direction.

ACKNOWLEDGMENT

This research is supported by the Brazilian agencies CAPES, CNPq (Grants: 408356/2018-9 and 305968/2018-1), FAPPR (Grant: 51435), and FAPERJ (Grant: 22520-7/2016).

REFERENCES

[1] L. Carvalho, A. Garcia, W. K. G. Assunção, R. Bonifácio, L. P. Tizzei, and T. E. Colanzi, "Extraction of configurable and reusable microservices from legacy systems: An exploratory study," in *23rd International Systems and Software Product Line Conference - Volume A*, ser. SPLC '19. New York, NY, USA: ACM, 2019, pp. 26–31.

[2] S. Newman, *Building microservices: designing fine-grained systems*. O'Reilly Media, Inc., 2015.

[3] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, 2017.

[4] M. A. Jarwar, S. Ali, M. G. Kibria, S. Kumar, and I. Chong, "Exploiting interoperable microservices in web objects enabled internet of things," in *Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2017, pp. 49–54.

[5] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[6] M. A. Naily, M. R. A. Setyautami, R. Muschevici, and A. Azurat, "A framework for modelling variable microservices as software product lines," in *Software Engineering and Formal Methods*, A. Cerone and M. Roveri, Eds. Springer, 2018, pp. 246–261.

[7] V. C. V. B. Segura, L. P. Tizzei, J. P. d. F. Ramirez, M. N. d. Santos, L. G. Azevedo, and R. F. d. G. Cerqueira, "WISE-SPL: Bringing multi-tenancy to the weather insights environment system," in *IEEE/ACM 5th International Workshop on Product Line Approaches in Software Engineering*, 2015, pp. 7–10.

[8] L. P. Tizzei, M. Nery, V. C. V. B. Segura, and R. F. G. Cerqueira, "Using microservices and software product line engineering to support reuse of evolving multi-tenant SaaS," in *21st International Systems and Software Product Line Conference*, ser. SPLC '17. New York, NY, USA: ACM, 2017, pp. 205–214.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[10] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," Gloriastrasse 35, CH-8092 Zurich, Switzerland, Tech. Rep. 103, 2001.

[11] B. Mohabbati, M. Asadi, D. Gašević, M. Hatala, and H. A. Müller, "Combining service-orientation and software product line engineering: A systematic mapping study," *Information and Software Technology*, vol. 55, no. 11, pp. 1845–1859, 2013.

[12] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions," in *VaMoS*, 2012, pp. 45–54.

[13] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "Reverse engineering feature models," in *ICSE*, 2011, pp. 461–470.

[14] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "On extracting feature models from sets of valid feature combinations," in *FASE*, 2013, pp. 53–67.

[15] L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed, "Feature model synthesis with genetic programming," in *SSBSE*, 2014, pp. 153–167.

[16] R. E. Lopez-Herrejon, J. A. Galindo, D. Benavides, S. Segura, and A. Egyed, "Reverse engineering feature models with evolutionary algorithms: An exploratory study," in *Search Based Software Engineering*, G. Fraser and J. Teixeira de Souza, Eds. Springer, 2012, pp. 168–182.

[17] W. K. Assunção, R. E. Lopez-Herrejon, L. Linsbauer, S. R. Vergilio, and A. Egyed, "Extracting variability-safe feature models from source code dependencies in system variants," in *Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO'15. New York, NY, USA: ACM, 2015, pp. 1303–1310.

[18] W. K. G. Assunção, R. E. Lopez-Herrejon, L. Linsbauer, S. R. Vergilio, and A. Egyed, "Multi-objective reverse engineering of variability-safe feature models based on code dependencies of system variants," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1763–1794, 2017.

[19] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *2017 IEEE International Conference on Web Services (ICWS)*, June 2017, pp. 524–531.

[20] J. Lee, D. Muthig, and M. Naab, "A feature-oriented approach for developing reusable product line assets of service-based systems," *Journal of Systems and Software*, vol. 83, no. 7, pp. 1123–1136, 2010.

[21] S. Khoshnevis and F. Shams, "Automating identification of services and their variability for product lines using NSGA-II," *Frontiers of Computer Science*, vol. 11, no. 3, pp. 444–464, Jun 2017.

[22] D. Benavides, S. Segura, P. Trinidad, and A. R. Cortés, "FAMA: Tooling a framework for the automated analysis of feature models," in *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, 2007, pp. 129–134.

[23] S. Segura, J. Galindo, D. Benavides, J. A. Parejo, and A. R. Cortés, "BeTTy: benchmarking and testing on the automated analysis of feature models," in *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, 2012, pp. 63–71.

[24] T. E. Colanzi, W. K. G. Assunção, P. R. Farah, S. R. Vergilio, and G. Guizzo, "A review of ten years of the symposium on search-based software engineering," in *Search-Based Software Engineering*, S. Nejati and G. Gay, Eds. Springer International Publishing, 2019, pp. 42–57.

[25] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 117–132, 2003.

[26] C. M. Fonseca, L. Paquete, and M. Lopez-Ibanez, "An improved dimension-sweep algorithm for the hypervolume indicator," in *IEEE Intern. Conference on Evolutionary Computation*, 2006, pp. 1157–1163.

[27] J. Cochrane and M. Zeleny, *Multiple Criteria Decision Making*. University of South Carolina Press, Columbia, 1973.

[28] R. Bergmann, J. Ludbrook, and W. P. J. M. Spooren, "Different Outcomes of the Wilcoxon-Mann-Whitney Test from Different Statistics Packages," *The American Statistician*, vol. 54, no. 1, pp. 72–77, 2000.

[29] A. Vargha and H. Delaney, "A critique and improvement of the cl common language effect size statistics of McGraw and Wong," *J. of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[30] A. Arcuri and L. Briand, "A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.

[31] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature-oriented software product lines*. Springer, 2016.