Scientific Research Publishing

Journal of Computer and Communications, 2020, 8, 114-126
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Learning to Support Derivation of Adaptable Products in Software Product Lines

Anissa Benlarabi*, Amal Khtira, Bouchra El Asri

IMS Team, ADMIR Laboratory, Rabat IT Center, ENSIAS, Mohammed V University Rabat, Morocco
Email: *a.benlarabi@gmail.com

How to cite this paper: Benlarabi, A., Khtira, A. and El Asri, B. (2020) Learning to Support Derivation of Adaptable Products in Software Product Lines. *Journal of Computer and Communications*, **8**, 114-126.
https://doi.org/10.4236/jcc.2020.84009

Received: February 17, 2020
Accepted: April 27, 2020
Published: April 30, 2020

Abstract

Software product line engineering is a large scale development paradigm based on mass production. It consists in building a common platform from which a set of products can be derived. Under the constraints of continuous evolution and costs optimization, the derivation process must be able to answer customers' requirements and provide adequate products in a short time without defects. However, this purpose cannot be achieved if the customer must wait for the change is implemented in the common platform. In this paper, we describe our work which proposes a framework to manage derivation of adaptable products. An adaptable product is obtained by deriving the most similar product from the common platform and changing it to support the new requirements. The aim of the framework is to give quick response to the customers need while the new requirements are being implemented in the common platform. The framework includes tools for processing natural language requirements, computing the similarity between products on the basis of their requirements, and the product adaptation effort measuring.

Keywords

Software Product Lines, Requirement, Similarity, Product Derivation, Natural Language Processing, Semantic Models

1. Introduction

The Software Product Line Engineering (SPLE) [1] is an approach that aims at creating individual software applications based on a core platform, while reducing the time-to-market and the cost of development. Many SPLE-related issues have been addressed both by researchers and practitioners, such as variability management, product derivation, reusability, etc. The focus of our work will be on products derivation in evolvable software product lines (SPL).

The classic derivation of products in software product lines consists in selecting features from the core platform and deriving a product using the existing software components. As a result, products cannot be adaptable to new requirements and cannot support new features unless the new requirements will be implemented in the level of the core platform. However, the evolution of the core platform can be costly and time consuming. Our study aims at finding a solution to the problem of deriving adaptable products while the implementation of new requirement is in progress, the objective is to provide a product that meets a customer's new requirements the earliest possible without waiting for their implementation at the platform level.

Our approach allows formalizing the representation of the existing requirements and also the existing products, the new requirements are expressed in the same way. Based on these representations, a set of algorithms are proposed to compare the new requirements with the existing ones in order to identify the most similar product. Therefore, the decision to adapt the selected product in response to the new requirements will be taken depending on the similarity level and the complexity of the adaptation task. An automated tool will be developed and verified using a case study in the mobile domain.

The remainder of the paper is organized as follows. Section 2 presents the background and the motivation. In Section 3, we define our approach. In Section 4, we describe the automated tool. Section 5 presents related works and Section 6 concludes the paper and gives the work perspectives.

2. Background and Motivation

In this section, we introduce the background of this study. We start by introducing the SPL engineering paradigm and the SPL evolution challenges, then we focus on the challenge of delivering products that respond to new business needs quickly until they are implemented in the SPL platform.

2.1. Software Product Lines

The software product line engineering consists on building a common platform for a set of products dedicated for a specific business domain [1]. The main advantage of the software product line engineering is the improvement of the productivity by reducing the time to market and the costs [2]. By using software product lines (SPL) customers will get products adapted to their needs and wishes at a reasonable price and a short time.

Software product line engineering encompasses two engineering processes. The first process is the domain engineering in which the common platform is developed including the requirements, the design, the realization and testing. The second process is application engineering in which the products of the family are derived from the common platform [2].

Evolution of software product lines is a complex task because it has two levels: the level of the products and the level of the platform, the requirements changes must be propagated in the two levels which requires a well understanding of the change and its impact. Due to the complexity of SPL evolution, much research was done to deal with SPL evolution issues, we present here some of them:

Modelling Evolution [3] [4] [5]: The objective of modelling evolution is to give a set of controlled and rigorous tasks to implement the change. Hence, the evolution is separated into a set of atomic operations independent from each other performed on a set of evolvable elements.

Traceability of the change [6] [7] [8]: The traceability approaches identify and trace existing links between the platform assets and also the links between the platform assets and the assets of the derived products. The goal is to make the evolution easier by determining how to propagate a change in a set on linked elements.

Co-evolution of domain and application engineering [9] [10]: this field of research focus on how the domain and the application processes evolve together and try to find changes that were implemented in one level in order propagate them to the other level.

Post-evolution verification [4]-[10]: The verification approaches study the risk of breaking the integrity of a software product line after changing it. They aim at improving the safety of software product line evolution by providing a toolset for checking if after refinement the software product line still generates well-formed products.

Model defects analysis: the approach dealing with model defects in SPL try to find defects caused by evolution such as inconsistency [5], incompleteness [11], ambiguity [11], and duplication [12] [13]. They also propose solutions to correct these defects. The majority of them focus on the feature level.

In our previous work [3], we proposed an approach to study the co-evolution between the platform and the derived products in order to correct the previous shortcomings in the evolution impact analysis tasks. In this paper we deal with evolution otherwise, we try to give quick response to a customer new requirements by selecting the most similar product to its needs and studying the cost of adapting it to support the requirements that are not implemented in the software product line platform. On the basis of the study, this solution can be adopted in agreement with the customer while waiting for the SPL change achievement.

2.2. Requirements Similarity Analysis

The concept requirement can be defined as follow:

"A condition or capability needed by a user to solve a problem or achieve an objective" [IEEE610.12-1990].

Requirements are the basis for every system, defining what the customers need from the system and also what the system must do in order to satisfy that need. They are generally expressed in natural language (NL), the language understood by all the software stakeholders. Requirement engineering [14] includes four principal activities: elicitation, documentation, validation and negotiation, and management [15]. Evolution of software product lines is expressed through requirements. Hence, the majority of works deal with evolution in the requirements or feature levels. In front of new requirements, the SPL analysts can choose one of the following scenarios to satisfy the customer needs:

- Implement the new needs at the domain level: customer must wait for the change implementation, then the desired product can be derived
- Study the possibility of adapting the existing products: in this case the most similar product is selected, we mean by the most similar product the one which supports the maximum of intended requirements. The product can be adapted and delivered while the new requirements are added in the domain level.

The response time is the inconvenience of the first scenario. For this reason we adopted in this paper the second scenario. In further sections, we describe our approach for similar product selection and adaptation in evolvable software product line.

3. Approach Description

The objective of our work is to provide the product that can satisfy customers even if the customer requirements are not completely supported by the software product line. The decision of adapting the products or implementing the modification in the platform is taken by stakeholders depending on the complexity of the modification. The solution proposed consists of a three-process framework. The first process consists in transforming the requirements into a formal presentation, the second process consists in selecting the most adequate product by comparing the new requirements with the existing products requirements, and the third process involves the measurement of the adaptation complexity. The overview of the framework is depicted in Figure 1. In what follows, we will explain in details the framework processes.

3.1. Process 1: Formalization of NL Requirements

The requirements related to a product are documented using natural language because it is the easiest way for customers to express their needs. This way of modeling does not enable the comparison of these requirements with the old

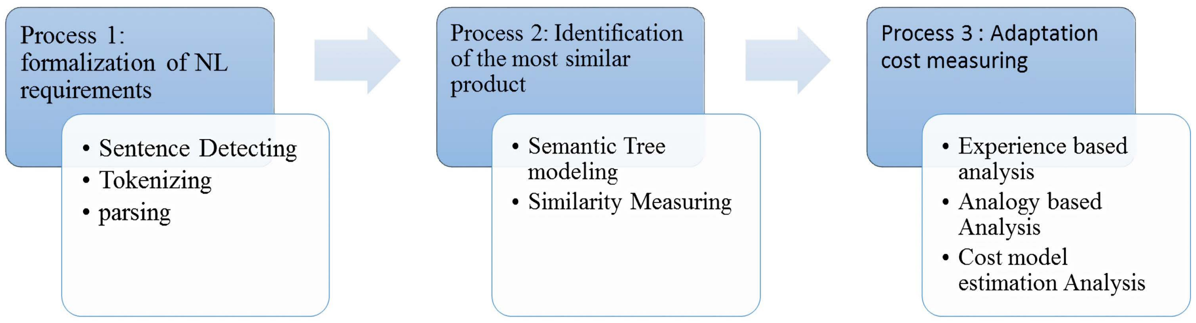
Process 1: formalization of NL requirements
• Sentence Detecting
• Tokenizing
• parsing
Process 2: Identification of the most similar product
• Semantic Tree modeling
• Similarity Measuring
Process 3 : Adaptation cost measuring
• Experience based analysis
• Analogy based Analysis
• Cost model estimation Analysis

Figure 1. Overview of the approach.

ones because the software product line platform support a great number of requirements. Thus, a formal representation is necessary.

We adopt Natural Language Processing (NLP) approach to transform a textual requirement into a tree-model. NLP is a technology of computer science whose aim is to convert text in natural language into a formal representation understandable by computers, it includes information extraction and syntax and semantic processing. However, a complete semantic analysis is still a subject for research.

The main NLP tasks are:

- Words extraction and text representation: aims at extracting terms (verbs, nouns, …) from a text and representing them into a vector.
- Part-Of-Speech Tagging (POS): aims at labeling each word with a unique tag that indicates its syntactic role.
- Chunking: aims at labeling segments of a sentence with syntactic constituents such as noun or verb phrase (NP or VP).
- Semantic Role Labeling (SRL): aims at giving a semantic role to a syntactic constituent of a sentence.

In our work we focus on the two first tasks, semantic analysis will be done in the second process. We formalize NL requirements using NLP techniques as depicted in Figure 2.

- Sentence Detector: enables the separation of sentences by putting each sentence in a different line.
- Tokenizer: divides each sentence into tokens (e.g. noun, verb, number).
- Parser: converts the sentence into a tree that represents the syntactic structure. Each word of the sentence is marked with a POS tagger (Part of-Speech tagger) that represents the role of this word in the English grammar.

3.2. Process 2: Identification of the Most Similar Product

Finding similar products consists in comparing the new requirements with the existing ones, the product which contains similar requirement is considered as a candidate product, the decision of taking the product among the proposed solutions depends on his similarity level and adaptation complexity.

In the literature the similarity between requirements is studied using the vector space model (VSM) [16]. In VSM, a requirement is represented as a vector of

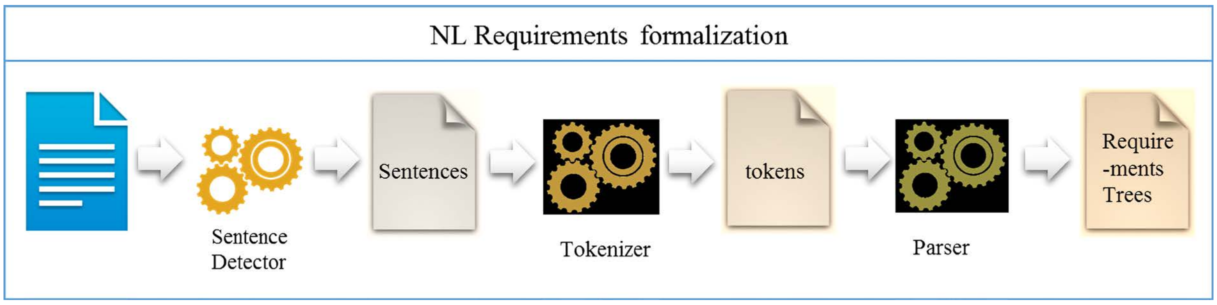

Figure 2. Process of NL requirement formalization.

identifiers or terms. This linear representation induces to many semantic analysis problems such as polysemy (same term that have different sense) and synonymy. For example we consider the two following sentences.

- S1: user must be able to collect prospects data by phone or from external databases.
- S2: the prospecting process starts by gathering information by telephone or from other information sources.

The two sentences S1 and S2 means the same thing, however if we consider their linear representations S1 (user, collect, prospects, data, phone, external, databases) and S2 (prospecting, process, start, gathering, information, telephone, other, information, sources). The two sentences are not similar. Hence, we adopted the hierarchical semantic representation of sentences as depicted in Figure 3.

The semantic hierarchy of terms in two tree models shows more clearly the similarity between the two sentences. In what follows we present the semantic tree model and the similarity calculation method using the tree model.

3.2.1. Semantic Tree Model

We define a semantic tree model as a hierarchical classification of a domain concept. In order to build such model for requirements we must have a taxonomy of the software product line business domain. A taxonomy is a set of concepts with PartOf or InstanceOf relations between them. In the software product lines domain, a taxonomy refers to a classification of domain concepts or features related with InstanceOf relations. The semantic tree of requirement is constructed on the basis of its syntax tree resulted from the requirements NLP processing and the SPL domain taxonomy. In the rest of this section we define the SPL domain taxonomy and the semantic tree model.

1) Taxonomy of SPL business domain

A taxonomy Tx is defined as a set of concepts and Instanceof/partOf relations between them $Tx = (Fx, Rx)$. In software product lines the concepts of taxonomy includes, features, business concepts, and measures. These elements are

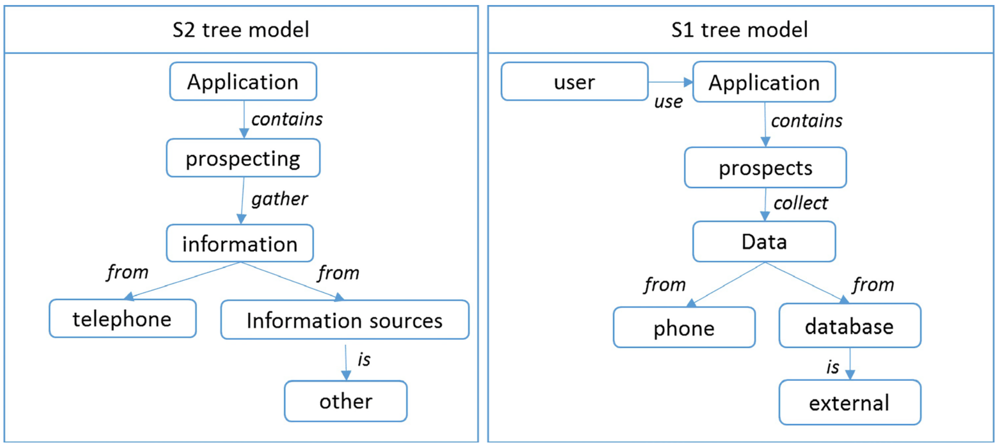

Figure 3. Tree models of the sentences $S1$ and $S2$.

classified with instanceof/partOf relations that allow for hierarchical classification of them. The taxonomy is used in requirements NLP processing specifically in terms extraction. Also, when constructing the semantic tree to locate a concepts among the others and collect their parents.

2) Semantic tree model

A semantic tree model is a graph composed of multiple nodes, each node represents a concept in its leaf and has one or more parents and one or more children. The edges of the tree depicts the relations between concepts. We consider the following representation of the Tree $T=(N,C,E,L)$

- N is the set of Nodes composing the tree $N=\{n_1,n_2,\cdots,n_k\}$
- C is the set concepts used in the leafs of nodes.
- E is a set of edges $E=\{e_1,\cdots,e_n\}$, in which $e=(n_i,n_j)$ is an ordered set representing an edge from node ni to node n_j.
- L is the set of values used in the edges to describe relation between nodes $L=\{l_1,\cdots,l_n\}$ in which $l_i \to e_i$

We define the following functions for the semantic tree:

- $P(n)$: Parent function returns the node parent of the node n.
- $L(n)$: Leaf function returns the concept presented in the leaf of node n.
- $Dfr(n)$: Distance from root function returns the distance between the node n and the root of the tree.

3.2.2. Similarity Measuring

At this level we consider our requirements formalized into a set of tree models. In this section we use the notation presented in the previous section in the different computations. The similarity between two semantic trees is defined as follows:

$$S(T1,T2)=\frac{1}{n}\sum_{i=1}^{n}\max\left(S(n_i,n_j)\right) \quad j\in 1\cdots m \tag{1}$$

$S(n_i,n_j)$ is the similarity between a node n_i in the tree $T1$ and a node n_j in the tree $T2$, n and m are respectively the cardinalities of $N1$ and $N2$. Each n_i is compared with all the nodes n_j of $N2$, then the most similar n_j is selected by the maximum function.

We compute the similarity between two nodes n_i and n_j as follows:

$$S(n_i,n_j)=\frac{\begin{array}{c}w_v * Sc(c_i,c_j)+w_p * Sp\left(P(n_i),P(n_j)\right)\\ +w_d * Sd\left(Dfr(n_i),Dfr(n_j)\right)+w_l * Sl\left(l_{i-1,i},l_{j-1,j}\right)\end{array}}{w_v+w_p+w_d+w_l} \tag{2}$$

We consider the following attributes of a node n_i in the tree $T1$:

c_i: The concept of the leaf of the node n_i

$P(n_i)$: The concept parent of the concept c_i

$Dfr(n_i)$: The distance of n_i from the root

$l_{i-1,i}$: The concept of the edge relating the node n_i to the node parent.

Similarity of each attribute is calculated separately and multiplied by a particular weight, we propose in Table 1 the attributes weight values ordered by their influences on the nodes similarity.

Table 1. Weights of attributes.

Symbol	Attribute	Weight
w_l	Concept of edge	0.1
w_d	Distance from root	0.2
w_p	Parent concept	0.3
w_v	Concept of leaf	0.4

Similarity function of each attribute is calculated as follows:

Similarity between concepts

$$Sc(c_i, c_j) = \begin{cases} 1 & \text{if } c_i = c_j \\ 0 & \text{if } c_i \neq c_j \text{ and they are not synonyms} \\ \alpha & \text{if } c_i \text{ and } c_j \text{ are synonyms} \end{cases}$$

Synonyms of c_i are retrieved from WordNet which gives a list of synonyms ordered by use frequency, the function $\rho(c_i)$ gives the number of synonyms found for a concept (c_i). Assuming that c_j is a synonym of c_i the function $\sigma(c_j)$ gives the order of c_j among the list of synonyms. The similarity between the two synonyms is:

$$\alpha = \left(\rho(c_i) - \sigma(c_j)\right) / \rho(c_i)$$

For example: we compute the similarity between the two concepts prospect and candidate. Prospect has 11 synonyms where candidate is the second one.

$$Sc(\text{prospect, candidate}) = \frac{11-2}{11} = 0.8$$

Similarity between *Dfr* of the two concepts

$$Sd\left(Dfr(n_i), Dfr(n_j)\right) = \begin{cases} 1 & \text{if } Dfr(n_i) = Dfr(n_j) \\ 0 & \text{if } Dfr(n_i) \neq Dfr(n_j) \end{cases}$$

Similarity between the parent concepts

$$Sp\left(P(n_i), P(n_j)\right) = Sc(n_{i-1}, n_{j-1})$$

Similarity between the edges concepts

$$Sl(l_{i-1,i}, l_{j-1,j}) = Sc(l_{i-1,i}, l_{j-1,j})$$

3.3. Process 3: Adaptation Cost Measuring

In this step we measure the cost of the similar product adaptation. The adaptation is the fact of implementing the requirement of intended product for which we did not found a similar requirement in the selected product. The cost can be calculated using the three following techniques:

- Expert opinion: the technical experts of the SPL system can estimate the cost of adaptation on the basis of their experience in SPL realization and evolution.
- Analogy: the cost of adaptation effort can be measured by comparing the im-

plemented operations with the operations to be implemented.

- Cost estimation models: such models estimate the cost on the basis of many factors, Constructive Cost Model COCOMO II [17] the most widely used model.

In our approach we propose to combine the three methods, we use the COCOMO II model to estimate the adaptation effort then we validate it with experts by analogy with other development projects.

4. Tool Support for Adaptable Product Derivation

In order to implement the proposed approach we propose a tool called APD-tool (adaptable product derivation tool). In this section we describe the tool features and its architecture.

4.1. APD-Tool Features

The proposed tool support the process illustrated in the approach description, the following functionalities support and implement the three processes:

- The import of textual description of the intended product composed of its NL requirements as a set of sentences.
- The processing of the textual description of the intended product in order to generate a graph of semantic Tree models from the NL requirements.
- The processing of existing products NL requirements to generate a graph of semantic tree models for each product.
- The measuring of similarity between the intended product and the existing ones.
- The selection of the most similar products.
- The estimation of adaptation effort costs for each similar products.

4.2. ADP-Tool Implementation

In order to implement the features described earlier, we designed the ADP-tool as depicted in Figure 4.

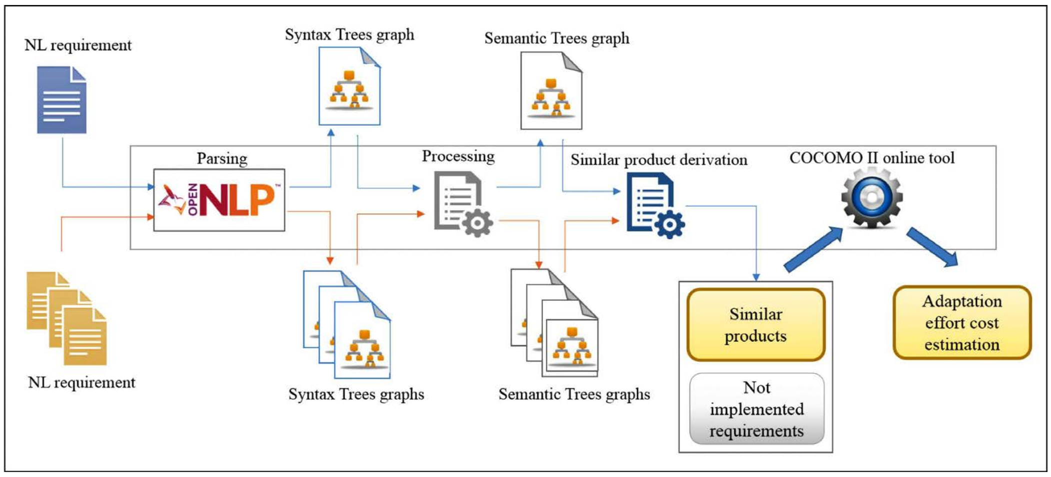

Figure 4. ADP-tool architecture.

The tool includes an interface developed with EclipseIDE which is a java Integrated Development Environment (IDE). The interface facilitates the selection of NL requirements files for users, and also show the resulted similar products with the requirement that must be implemented to adapt them to the customer need.

The syntax processing of the textual NL requirements is performed with the openNLP tool [18], which is a machine learning based toolkit for the processing of natural language text. It supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and co-reference resolution. It also includes an evaluation tool that measures the precision of entity recognition and provides information about the accuracy of the used model.

The syntax trees graph generated using the OpenNLP tool are processed then by an algorithm in order to create the semantic graph from the intended product and the existing derived products. We suppose that each NL requirement is expressed in an independent sentence. This algorithm creates a tree for each NL requirement sentence using their syntax graph and the taxonomy of the SPL domain.

The similar products are selected using the algorithm of similar product derivation, which performs the computation of the trees similarity as presented in Section 3.2.2. Each NL requirement Tree is compared with all the requirements trees of a candidate product, then the most similar requirement is selected. The similarity between the two graphs is the average of their trees similarities.

When the similar products are selected, their adaptation to include the not implemented requirements is then studied. The adaptation cost effort is calculated using COCOMO II model. We can use an online tool which allows for applying the model and calculate the cost.

5. Related Works

In the literature many works were proposed to deal with requirements similarity, however there is no approaches dealing with requirement similarity analysis in the SPL field.

The authors in [19] propose an approach to identify a hierarchy of a set of software products. The hierarchy provides a classification scheme associated with a set of attributes, a non-classified item is included in a specific location based probabilistic matching among the attributes of products in the hierarchies, and the known attributes of the item. Such hierarchy helps identifying the similarity level between products of a specific domain.

A new configuration approach for software product lines based on the user preferences [20]. Instead of selecting features from the domain features to satisfy the user need, a new constraint is added in the configuration process which is user preferences. To allow more fine-grained user customization at runtime, user profiles are introduced which represent the desires of users. In particular, this means the desire to select or deselect features under certain circumstances.

Another approach was presented in [21], which speedy reaction to new context conditions and better support for evolution of Wireless Sensor and Actuator Networks, WSANs. They focus on those WSAN features that are susceptible to being managed by a dynamic software product lines (DSPL) approach. They propose a tool to reconfigure the products DSPL of WSAN at runtime. The reconfiguration involves changes at execution time in specific parts of code to give a flexible solution.

A concept similar to our work is the context variability [22], it consists in combining domain features with contextual information for product line derivation. According to contextual information the products to derive is determined, and additional contextual changes are incorporated at execution time.

Finally, works that deal with reconfiguration of products in a SPL at derivation time focus on contextual variability especially the environmental changes and the resources constraints. In our work we focus on functional requirements evolution because of their priority and the frequency of their changes.

6. Conclusion

In this paper, we introduced our framework for adaptable products derivation, which studies the similarity between an intended product and the existing products in software product lines based on their requirements expressed in natural language. Our goal is to optimize the response time to customers business needs, instead of adopting classical approach which consists in waiting the implementation of the new requirements in the common platform and then building the relevant product, our solution allows for a new approach by selecting the most similar product from the existing ones and computing the cost of its adaptation. The stakeholders can select one approach, depending on the adaptation effort cost. So far, we have proposed a tool that automates the three processes of our framework that are: 1) transforming the natural language requirements into a semantic trees, 2) selecting the most similar product by calculating the similarity function between the intended product and the existing ones, 3) measuring the cost of adapting the selected product. In our future work, we intend to implement the automated tool and validate the results through two cases studies, one in the mobile field and the other in the customer relationship management field.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Pohl, K., Böckle, G. and van Der Linden, F.J. (2005) Software Product Line Engineering: Foundations, Principles and Techniques. Springer Science & Business Media, New York. https://doi.org/10.1007/3-540-28901-1

[2] Northrop, L. (2002) Software Product Lines: Practices and Patterns. Addison-Wesley, New York.

[3] Benlarabi, A., Khtira, A. and El Asri, B. (2015) A Co-Evolution Analysis for Software Product Lines: An Approach Based on Evolutionary Trees. *International Journal of Applied Evolutionary Computation*, **6**, 9-32. https://doi.org/10.4018/IJAEC.2015070102

[4] Ferreira, F., Borba, P., Soares, G. and Gheyi, R. (2012) Making Software Product Line Evolution Safer. 2012 *Sixth Brazilian Symposium on Software Components, Architectures and Reuse*, Natal, Brazil, 23-28 September 2012, 21-30. https://doi.org/10.1109/SBCARS.2012.18

[5] Nuseibeh, B. (1996) To Be and Not to Be: On Managing Inconsistency in Software Development. *Proceedings of the* 8*th International Workshop on Software Specification and Design*, Schloss Velen, Germany, 22-23 March 1996, 164-169. https://doi.org/10.1109/IWSSD.1996.501161

[6] Goknil, A., Kurtev, I., van den Berg, K. and Veldhuis, J.W. (2011) Semantics of Trace Relations in Requirements Models for Consistency Checking and Inferencing. *Software & Systems Modeling*, **10**, 31-54. https://doi.org/10.1007/s10270-009-0142-3

[7] Anquetil, N., Kulesza, U., Mitschke, R., Moreira, A., Royer, J.C., Rummler, A. and Sousa, A. (2010) A Model-Driven Traceability Framework for Software Product Lines. *Software & Systems Modeling*, **9**, 427-451. https://doi.org/10.1007/s10270-009-0120-9

[8] Passos, L., Czarnecki, K., Apel, S., Wąsowski, A., Kästner, C. and Guo, J. (2013) Feature-Oriented Software Evolution. In: *Proceedings of the Seventh International Workshop on Variability Modelling of Software-Intensive Systems*, ACM, New York, 17. https://doi.org/10.1145/2430502.2430526

[9] Benlarabi, A., El Asri, B. and Khtira, A. (2014) A Co-Evolution Model for Software Product Lines: An Approach Based on Evolutionary Trees. 2014 *Second World Conference on Complex Systems* (*WCCS*), Agadir, Morocco, 10-12 November 2014, 140-145. https://doi.org/10.1109/ICoCS.2014.7060991

[10] Apel, S., Rhein, A.V., Wendler, P., Größlinger, A. and Beyer, D. (2013) Strategies for Product-Line Verification: Case Studies and Experiments. In: *Proceedings of the* 2013 *International Conference on Software Engineering*, IEEE Press, New York, 482-491. https://doi.org/10.1109/ICSE.2013.6606594

[11] Lami, G., Gnesi, S., Fabbrini, F., Fusani, M. and Trentanni, G. (2004) An Automatic Tool for the Analysis of Natural Language Requirements. Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Setiembre.

[12] Thomas, D. and Hunt, A. (2000) The Pragmatic Programmer.

[13] Khtira, A., Benlarabi, A. and El Asri, B. (2015) A Tool Support for Automatic Detection of Duplicate Features during Software Product Lines Evolution. *International Journal of Computer Science Issues*, **12**, No. 4.

[14] Hull, E., Jackson, K. and Dick, J. (2010) Requirements Engineering. Springer Science & Business Media, New York.

[15] Pohl, K. (2016) Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam-Foundation Level-IREB Compliant. Rocky Nook, Inc., New York.

[16] Turney, P.D. and Pantel, P. (2010) From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research*, **37**, 141-188. https://doi.org/10.1613/jair.2934

[17] Boehm, B., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E. and Steece, B. (2000) Cost Estimation with COCOMO II. Prentice-Hall, Upper Saddle River,

NJ.

[18] The Apache Software Foundation (2015) OpenNLP. https://opennlp.apache.org

[19] Hunt, H.D., West, J.R., Gibbs Jr., M.A., Griglione, B.M., Hudson, G.D.N., Basilico, A., Yusko, J.A., *et al.* (2016) U.S. Patent No. 9,262,503. U.S. Patent and Trademark Office, Washington DC.

[20] Nieke, M., Mauro, J., Seidl, C. and Yu, I.C. (2016) User Profiles for Context-Aware Reconfiguration in Software Product Lines. In: *International Symposium on Leveraging Applications of Formal Methods*, Springer, Cham, 563-578. https://doi.org/10.1007/978-3-319-47169-3_44

[21] García Hernando, A.B., Ortiz Ortiz, Ó., Capilla Sevilla, R., Bosch, J. and Hinchey, M. (2012) Runtime Variability for Dynamic Reconfiguration in Wireless Sensor Network Product Lines.

[22] Sboui, T., Ayed, M.B. and Alimi, A.M. (2017) Addressing Context-Awareness in User Interface Software Product Lines (UI-SPL) Approaches. In: *Human Centered Software Product Lines*, Springer, Cham, 131-152. https://doi.org/10.1007/978-3-319-60947-8_5