IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 17, 2019, accepted May 7, 2019, date of publication May 22, 2019, date of current version June 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2918469

Selection of Software Product Line Implementation Components Using Recommender Systems: An Application to Wordpress

JORGE RODAS-SILVA[1], JOSÉ A. GALINDO[2], JORGE GARCÍA-GUTIÉRREZ[2], AND DAVID BENAVIDES[2]

[1]Facultad de Ciencias de la Ingeniería, University of Milagro, Milagro 091050, Ecuador
[2]Departamento Lenguajes y Sistemas Informáticos, University of Seville, 41012 Seville, Spain

Corresponding author: Jorge Rodas-Silva (jrodass@unemi.edu.ec)

This work was supported in part by the EU FEDER Program of the MINECO projects OPHELIA (RTI2018-101204-B-C22), in part by the Big Time-Aware Data: Análisis de Datos Masivos Indexados en el Tiempo. Reglas y Clustering under Grant TIN2014-55894-C2-1-R, in part by the Big Data Streaming: Análisis de Datos Masivos Continuos. Modelos Descriptivos under Grant TIN2017-88209-C2-2-R, in part by the Juan de la Cierva postdoctoral program, in part by the TASOVA network under Grant MCIU-AEI TIN2017-90644-REDT, in part by the Junta de Andalucia METAMORFOSIS Project, and in part by the University of Milagro with its Scholarship Program.

ABSTRACT In software products line (SPL), there may be features which can be implemented by different components, which means there are several implementations for the same feature. In this context, the selection of the best components set to implement a given configuration is a challenging task due to the high number of combinations and options which could be selected. In certain scenarios, it is possible to find information associated with the components which could help in this selection task, such as user ratings. In this paper, we introduce a component-based recommender system, called (REcommender System that suggests implementation Components from selecteD fEatures), which uses information associated with the implementation components to make recommendations in the domain of the SPL configuration. We also provide a RESDEC reference implementation that supports collaborative-based and content-based filtering algorithms to recommend (*i.e.*, implementation components) regarding WordPress-based websites configuration. The empirical results, on a knowledge base with 680 plugins and 187 000 ratings by 116 000 users, show promising results. Concretely, this indicates that it is possible to guide the user throughout the implementation components selection with a margin of error smaller than 13% according to our evaluation.

INDEX TERMS Feature models, implementation components, recommender systems, software product line, wordpress.

I. INTRODUCTION

A *Software Product Line* (SPL) is defined as a set of software-intensive systems that share a set of common features that can be customized according to the specific needs of the stakeholders in a particular context [1], [2]. For the management of an SPL, models are used to represent all the possible products that can be derived from it. The most popular models used for this purpose, are called *feature models* [3]. Feature models graphically represent the common and variable features in an SPL. Understanding the impact of feature selections of an SPL is a costly and error-prone activity due to a combinatorial explosion of possible combinations of features.

To overcome this problem, research has proposed the use of the so-called *Automated Analysis of Feature Models* [4], [5] which, using computer-aided techniques and tools, allow configuration space management. However, the problem becomes more complex when for each feature of a configuration, there is more than one component that implements it.

Consider the example of eCommerce websites development in WordPress. When a website is configured in this

The associate editor coordinating the review of this manuscript and approving it for publication was Fabio Gasparetti.

platform, each feature can be implemented by a set of plugins. If it is necessary to implement the *online payment* feature, there may be more than one plugin that provides this functionality. Selecting the plugins that best adapt to this requirement, considering the variety of plugins (WordPress is comprised of more than 55,000 plugins,)[1] can be a critical, tedious, error–prone and time-consuming activity. In this example, the eCommerce website defines the SPL domain, the website functionalities represent the features, and the plugins set represents the alternative components set to implement a certain feature.

A possible way to select appropriate components is to analyze information associated with them. This information is generally provided by users through reviews, reports of errors or ratings about the products in use. As an example, in website configurations, user ratings could be an indicator to determine how well they worked when implemented on a certain website and, at the same time, it can be a guide to other users in similar developments.

The information provided by users through ratings can be exploited by *Recommender Systems*. Recommender Systems have been successfully applied in various scenarios such as online stores (Amazon), over-the-top media services provider (Netflix), among others [6]. Recommender systems suggest personalized products and services to users according to their preferences and tastes.

There are several works in the literature that use recommender techniques to support the SPL configuration process. These studies address configuration from different perspectives and contexts. For example, feature model structural properties [7]–[11] in which the use of recommendation techniques are used for the configurations selection and prioritization. Also, using quality attributes [12]–[15] which use contextual information of features to select optimal configurations and offer a more personalized product to the user.

These studies do not research the specific components selection scenario to implement features. In other words, in product configuration environments, there is no effective solution using recommender systems to efficiently assist users to find suitable components to implement features.

To address this challenge, we propose a component-based recommender system, called RESDEC (REcommender System that suggest implementation Components from selecteD fEatures). RESDEC aims to take advantage from an information repository generated by users to efficiently search for appropriate components to implement features in a specific context. To this end, we adapt a set of recommendation algorithms, based on collaborative and content filtering techniques. Concretely, we present the following contributions:

1) Modelling the implementation components selection problem as a recommendation task using collaborative–based and content–based filtering, addressing three common scenarios that arise during the configuration of products.
2) A prototype of a component-based recommender system tool ready to use and extend to other environments where it is necessary to configure the features of a product from the implementation components selection.
3) An empirical evaluation of our approach in an scenario of eCommerce website using Wordpress. The evaluation is based on data from 116,000 users, 680 plugins, and 187,000 ratings. Results show promising values with a margin of error smaller than 13% according to our evaluation.

The rest of the paper is structured as follows: Section II introduces a brief background on SPLs and recommender systems. Section III, present the motivation for the problem and illustrates it through an example. Section IV provides an overview of our proposed solution. Section V describes de design and results of the performed experiments. Section VI discusses the related work. Section VII outlines directions for future work. Finally, Section VIII concludes the paper.

II. PRELIMINARIES

In this section, we present all the required information to make this paper self-contained. Initially, we introduce *Software Product Lines* (SPL) with a brief overview on feature models and product configuration. Furthermore, we present the basic concepts about non-personalized and personalized *Recommender Systems* with an overview of the algorithms used in this work.

A. SOFTWARE PRODUCT LINES

According to Clements and Northrop [16], an SPL is a set of software-intensive systems that share a set of common features that meet the specific needs of a market segment. A feature is defined as a software functionality. In 1990, the FODA (*Feature-Oriented Domain Analysis*) [3] introduced the concept of *feature models*, which remained as one of the main research areas in SPL engineering. Feature models allow to graphically represent all the valid configurations of an SPL in terms of features and relationships in a compact manner. A graphical illustration, known as a *feature diagram*, is shown in Fig. 1. In feature diagrams, features are represented by boxes and relationships by edges. Features can be abstract or concrete [17]. A feature is *abstract*, if it is not mapped to any implementation artifact, and it is *non-abstract or concrete*, when it is assigned to at least one implementation artifact. Batory *et al.* [18] classifies relationships into three groups: ''and-group'', ''or-group'', and ''alternative-group''. *And-group* can be mandatory or optional. A relationship is *mandatory* when a child feature has a mandatory relationship with its parent and it is included in all the products in which its parent feature appears; and it is *optional*, when a child feature has an optional relationship with its parent and can optionally be included in all the products in which its parent feature appears.

According to the model presented in Fig.1 an eCommerce website should implement a *catalog of products, a payment*

[1] WordPress Website: https://wordpress.org/plugins/

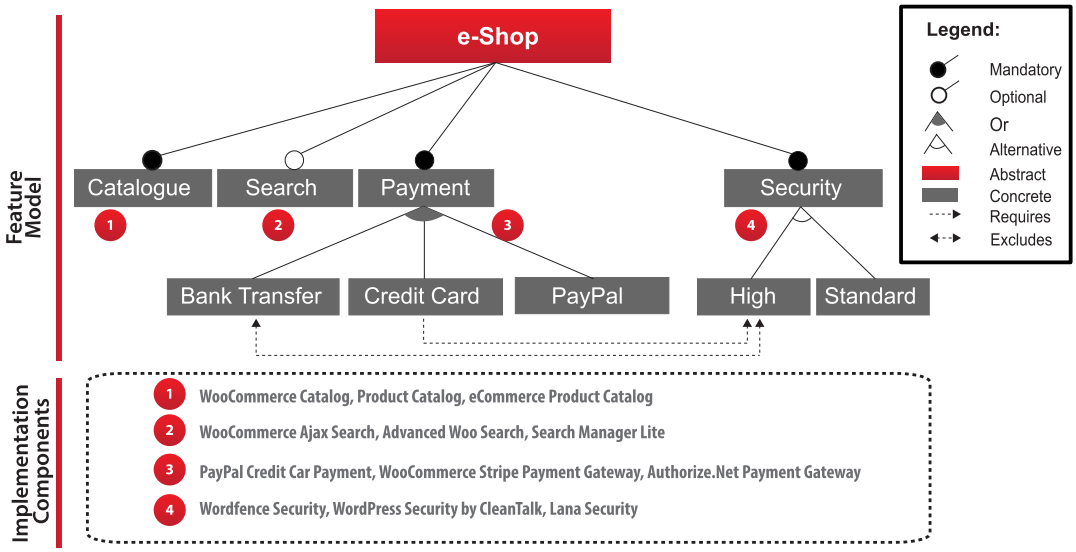

FIGURE 1. A simplified feature model describing an eCommerce-based website. Below to the feature model representation, we show the list of components that could implement each feature.

module, security policies and could optionally include a *search tool*.

In an *or-group* when a parent feature is selected, at least a child sub-feature must be selected. In the payment module of an eCommerce website, a combination of the features *Bank Transfer*, *Credit Card*, *and PayPal* could be activated. Finally, in an *alternative-group*, when a parent feature is selected, only a child sub-feature can be selected. For example, the eCommerce website must have one of the following active security policies: *high or standard*; but not both.

In addition to the parental relationships between features, a feature model can also contain restrictions between non-connected features [4]. These restrictions can be of two types: *requires* and *excludes*. A restriction is *requires* when an A feature requires a B feature, that is, the inclusion of A also implies the inclusion of B within the product configuration. For example, for the eCommerce website in Figure 1, implementing *credit card* payment function requires *high security* policies. Otherwise, it is *excludes* when a feature A excludes a feature B, that is, both features cannot be part of the same product. In the case of the eCommerce website, the policies of *high security* and payments by *bank transfer* are incompatible, since the activation of high security policies requires the use of a *credit card* or *PayPal* payment.

A set of selected features derived from the model will create a product configuration. A product configuration is valid if it satisfies all the feature model constraints. There are several configurators in the SPL literature to guide users into a valid configuration by using visualization mechanisms [19]. Moreover, recent approaches suggest the use of recommender techniques to customize configurations to users [8]–[10], [13]–[15]. In the work presented by Pereira *et al.* [13], the authors extend a state-of-the-art configurator with personalized collaborative-based recommender techniques. Despite the efforts made so far in the SPL literature, we do not know any SPL configurator to support the component-based configuration process. In the component-based configuration scenario, a product configuration contains a set of necessary components to implement each feature. This set of components are called *implementation components* [20].

As an example, *catalog, search, payment and security* features of the eCommerce website presented in Figure 1 can be implemented using the list of components shown at the bottom of the figure. It is noted that, several features can be implemented by the same component. In a specific case, if we want to implement the *catalog* feature identified with 1, we could have more than one selection alternative to implement it. As an illustration in the Figure 1, we present a list of three components in which the user could choose one. However, in the practice we can find a wide variety of components available for selection which could also implement the *catalog feature*, which makes critical the implementation process of the feature.

In this context, we conclude that component-based configuration process seems to be equally (or even more) challenging than the usual feature-based configuration process. In section III, we present the main challenges faced in this scenario and illustrate it using an example.

B. RECOMMENDER SYSTEMS

A recommender system (RS) is defined as a system that provides users with a series of suggestions in a personalized way according to their tastes or preferences [21]. There are successful practical examples of implementing recommender systems in actual scenarios, such as *Amazon* [22] and *Netflix* [23]. Recommender systems are mainly divided [24] into two groups: *collaborative-based* and *content-based*

recommender systems. As follows, we will provide more details on these techniques, emphasizing the algorithms that we will use in the present work.

1) COLLABORATIVE-BASED RECOMMENDER SYSTEMS

The systems based on collaborative filtering techniques [22], [25], also known as personalized recommender systems, are based on the analysis of user profiles, where recommendations are generated according to the tastes of users with similar preferences. For example, on *Netflix*, a user who has rated a series of movies could receive recommendations from other users who have also rated the same movies in a similar way or at least a large part of them, which we call a group of users with similar interests.

One of the main challenges of collaborative filtering systems is how to recommend to new or inexperienced users, which means, users who have never used the system, so there is no record of their interests. The lack of experience of these users makes it difficult to find relevant results (similar users or items) that fit their profiles. This case is defined in recommender systems as *Cold Start*. Non-personalized recommender systems have been introduced in the literature to solve cold-start problems [26]. This recommendation technique calculates the average rating ($\bar{r}_{p_j}$) of each item (p_j) from the users who have rated it (U_{p_j}) (see Equation 1). Then, the best rated items are selected from Equation 2 to build the recommendation of the new user.

$$\bar{r}_{p_j} \leftarrow \frac{\sum_{u_i \in U_{p_j}} r_{u_i}}{|U_{p_j}|}; j = 1..n \tag{1}$$

$$P_{u_k} \leftarrow max_t(\bar{r}_{p_j}) \tag{2}$$

As it is observed, this technique is very general, so it does not customize the recommendations.

A most interesting case in the collaborative filtering scenario is to make recommendations to users who have experience in the system, which means that have issued some kind of ratings for used items, so the system uses the records to find other users with similar interests. Next, we will describe two collaborative-filtering techniques used to build recommendations to experienced users.

Memory-based collaborative-filtering: Memory-based collaborative filtering algorithms [27] are characterized by employing the entire matrix of ratings to generate predictions (*i.e.*, estimate how a user would rate each item). In these algorithms, each user is part of a group of people with similar interests that is known as ‘‘neighborhood". From the identified neighborhoods, preferences can be combined to make predictions. The most commonly used approaches in this category according to the literature are the so-called *neighbourhood-based collaborative filtering (kNN-CF)*. KNN-CF algorithms [28] use statistical techniques to find neighbors with a ratings record similar to the active user ratings (*i.e.*, user for whom recommendations are done). When the nearest neighbors are found, their preferences are combined to create a list of recommendations for an active user. Two well-known KNN-CF algorithms are: *user-user KNN* and *item-item KNN*.

- **User-User KNN:** This algorithm [29] uses the experience of other users to build recommendations to an active user. The input of the system is a matrix of ratings (M). Ratings are collected in advance by measuring the relevance of the items by users. The similarity between users is established by the *Pearson Correlation Coefficient* (PCC) [30]. In Equation 3, we compute the similarity between an active user u_k and any other user of the system u_i.

$$s_{u_k,u_i} = \frac{\sum_{p \epsilon P}(r_{u_k,p_j} - \bar{r}_{u_k})(r_{u_i,p_j} - \bar{r}_{u_i})}{\sqrt{\sum_{p \epsilon P}(r_{u_k,p_j} - \bar{r}_{u_k})^2}\sqrt{\sum_{p \epsilon P}(r_{u_i,p} - \bar{r}_{u_i})^2}} \tag{3}$$

  where:

  - s_{u_k,u_i} represents the similarity between the user k and the user i for $k \neq i$.
  - P is the set of items rated by users u_k and u_i.
  - r_{u_k,p_j} and r_{u_i,p_j} is the rating on the item p_j issued by the user u_k and u_i respectively.
  - $\bar{r}_{u_k}$ and $\bar{r}_{u_i}$ is the average rating on all items also rated by u_k and u_i.

  Once the similarity between the users is established, the algorithm uses these results in Equation 4 to predict the relevance that the user u_k would give to those items p not yet rated.

$$r_{u_k,p} = \bar{r}_{u_k} + \frac{\sum_{u \in U}(r_{u_i,p} - \bar{r}_{u_i})S_{u_k,u_i}}{\sum_{u \in U} S_{u_k,u_i}} \tag{4}$$

  where:

  - $r_{u_k,p}$ is the possible rating that would give u_k to the item p.
  - S_{u_k,u_i} is the similarity between users k and i (result of Equation 3).
  - U is the set of users (more) similar to u_k. This set of users varies depending on the user population. In this case the top-10 of the best ratings has been used.

- **Item-Item KNN:** This algorithm, unlike the previous one, generates the recommendations based on the similarities between the items [27] rated by an active user. The similarity between items is also calculated through the PCC (i.e., similar to Equation 3). Equation 5 describes this process:

$$s_{p_i,p_j} = \frac{\sum_{u \epsilon U}(r_{u,p_i} - \bar{r}_{p_i})(r_{u,p_j} - \bar{r}_{p_j})}{\sqrt{\sum_{u \epsilon U}(r_{u,p_i} - \bar{r}_{p_i})^2}\sqrt{\sum_{u \epsilon U}(r_{u,p_j} - \bar{r}_{p_j})^2}} \tag{5}$$

  where:

  - s_{p_i,p_j} determines the similarity between the items p_i and p_j.

- U is the set of all the users who have rated both the item p_i and p_j.
- r_{u,p_i} is the rating of user u on the item p_i.
- $\bar{r}_{p_i}$ is the average rating on the item p_i.

Once the similarity between the items is established, the algorithm predicts the rating of the user u_k for an item p_i not yet rated, using the Equation 6.

$$r_{u_k,p_i} = \frac{\sum_{p_j \in P} r_{u_k,p_j} S_{p_i,p_j}}{\sum_{p_j \in P} |S_{p_i,p_j}|} \tag{6}$$

where:

- S_{p_i,p_j} is the similarity between the items p_i and p_j (result of Equation 4).
- P represents the set of items more similar to the item p_i.

Model-based collaborative filtering: The ratings matrix is often very large and sparse. For this reason, model-based collaborative filtering algorithms [27] use knowledge base reduction techniques which aim to decompose the matrix into smaller ones, which reflect the common characteristics of the original matrix. These algorithms create a model through which matrices of smaller dimensions are built. These matrices represent the affinity degree between users and items. Thus, they may allow the system to recognize patterns that may be hidden in the dataset.

Unlike the algorithms based on memory, model-based collaborative filtering does not use the whole set of items and users to make predictions. Previously, it performs a pre-filtering process to create groups or user segments based on their common interests. From these segments, it establishes the recommendations.

In this work, we use the *Matrix factorization (MF)* algorithm [31], characterized by decomposing the matrix of ratings in n sub-matrices. This algorithm, according to the literature, is designed to process large volumes of data, achieving good scalability, more accurate predictions and flexibility in the model. More specifically, we have used SVD for MF in our implementation.

- **Singular Value Decomposition (SVD).** This algorithm [32] decomposes the rating matrix (M) into three matrices (U, V, S). The first one is an orthogonal matrix $U_{n\times n}$ that represents the relationships between users. The second one is an orthogonal matrix $V^t_{m\times m}$ that determines the relationships between the features. The third one is a diagonal matrix $S_{n\times M}$ that establishes the relationship between both matrices.
  To improve efficiency, the algorithm applies the Eckart-Young theorem [33] that obtains an approximation with only k factors ($k < n$), so that the matrices would remain as $U_{n\times k}$, $S_{k\times k}$ and $V^t_{k\times m}$, as it is shown in Equation 7.

$$SVD(M_{1m\times n}) \simeq U_{m\times k} \times S_{k\times k} \times V^T_{k\times n} \tag{7}$$

  After this process, calculating the prediction of the user u_k on the item p is reduced by multiplying the k-th row vector of the matrix U (*i.e.*, $U(u_k)$) with the matrix S and the $p-th$ vector column of the matrix V^t, ($V^t(p)$) as shown in the Equation 8.

$$ru_k, p = \bar{r}_{u_k} + U(u_k) \times S \times V^T(p) \tag{8}$$

Collaborative filtering algorithms use different equations to estimate the ratings ($r_{u_k,p}$) of an active user on a set of target items (see Equations 1, 4, 6, and 8). These algorithms returns the k the best rated items.

2) CONTENT-BASED RECOMMENDER SYSTEMS

Content-based recommender systems [34] make recommendations based on the characteristics of the items. Without the need to use information from other users. They are generally used for information retrieval, such as search engines. In this work, we use TF-IDF (Term frequency – Inverse document frequency) algorithm to the SPL configuration domain.

- **TF-IDF:** The TF-IDF algorithm [35] is commonly used to perform customized searches by internet search engines. It is characterized by being able to find the *local weight* and the *global weight* in a collection of documents that are being analyzed. The *local weight* is known as TF (*Term Frequency*) and specifies the number of times a word is repeated within a document; while the *global weight*, known as IDF (*Inverse Document Frequency*), indicates the number of documents in which that word appears at least once. The number of TF and IDF occurrences for each document determines the elements to recommend.
  To adapt this algorithm to the SPL configuration domain, we use a user rating matrix M and a binary matrix N that relates the items with their features. The vector v_u describes the ratings of each user and a binary vector v_p determines the profile of each item p. The vector v_u represents the frequency in which each feature appears in the item rated by the user u, and the vector v_p determines the presence or not of each characteristic in the items p (obtained by each row of the matrix N). Vectors v_u and v_p have the same dimension, determined by the number of features f in the matrix N.
  Once we attributed the values to the vectors, the algorithm uses the TF-IDF strategy to obtain a weight for each characteristic, penalizing those that are not very similar and rewarding the most distinctive ones. Equation 9 presents the way to calculate the weighting of each feature f_i

$$w_{f_i} = F(f_i, u_k).log\left(\frac{n}{IF(f_i)}\right) \tag{9}$$

  where $F(f_i, u_k)$ represents the frequency which the characteristic f_i appears in the items rated by the user u_k and represents $IF(f_i)$ the inverse frequency or number of times the same characteristic appears but in the items that have not yet been rated by u_k.
  Then, to recommend items to the user u_k, the algorithm calculates the cosine similarity (see Equation10) between the user profile (v_{u_k}) and the profile of the items

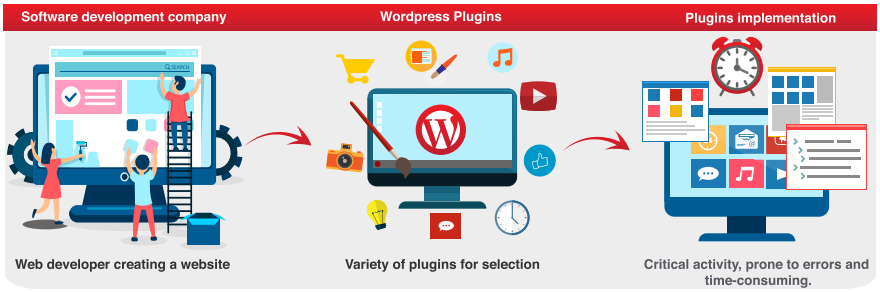

FIGURE 2. Motivation scenario inspired on WordPress.

not rated by the user ($v_p, p \in \overline{P}$). In this computation, we use the weight vector calculated according to Equation 9.

$$S_{cos}(v_{u_k}, v_p, w) = \frac{\sum_{i=1}^{t} v_{u_k}(i) * v_p(i) * w(i)}{\sqrt{\sum_{i=1}^{t}(v_{u_k}(i))^2} * \sqrt{\sum_{i=1}^{t}(v_p(i))^2}} \tag{10}$$

Finally, the algorithm recommends the k items with higher similarity with the user.

III. MOTIVATING SCENARIO

The way in which software is developed today differs extraordinarily from the way it was done 20 years ago. For example, there are multiple scenarios in which it is no longer necessary to develop applications from scratch. The great flexibility shown by the so-called *configurable platforms*, greatly facilitate the creation of new software. In general terms, it is enough to use a pre-configured architecture, and add the functionalities required in the form of existing components to obtain as a result, a solution if not definitive, close enough to the one desired. Actual examples of these configurable platforms, in the scenario of web development, are the content managers (CMS) WordPress, Joomla and Drupal [36].

However, although the process of creating new software has been benefited through the availability of functionalities in the form of components, obtaining proper configuration that meets a set of given requirements is a complex activity. Mainly, due to the huge number of options (concrete variants of application), that can be generated from the combination of existing components. A similar scenario occurs in the domain of SPL configuration. In SPL, feature models capture the common and variable aspects of a family of similar products, which allow the generation of different software variants. These variants may later be customized with the inclusion of specific implementation components that add a functionality to existing features defined in the feature model.

For a better understanding, Figure 2 shows an example of a configurable platform, in which a web developer faces the difficult task of selecting plugins (*i.e.*, implementation components) to implement the *online payments* feature in an eCommerce website. To implement this feature, it is possible to choose among several available options of plugins. The typical way to search for plugins on configurable platforms (including online software repositories) is to perform a search for keywords frequently related to the functionality desired to be added in the application. Therefore, understanding the correlation between feature selections and plugins is important for decision makers to be able to select a plugin that best suits their needs and expectation.

This task takes time and it does not always guarantee that the selected components are the most adequate (in terms of quality) for the required application. To the best of our knowledge in configurable development environments, there are no effective solutions to assist the developer in the task of finding suitable implementation components.

Our focus in this paper is to guide users through this process by employing techniques from recommender systems.

Next, we motivate three scenarios that can occur frequently in the process of setting up a website in WordPress and may benefit from a recommender system.

1) A common case is a developer who for the first time start a web development in WordPress. Normally, the developer starts building his new website and has a default structure (*e.g.*, of the own configurable platform chosen to implement his solution). This structure is generally an invariable kernel of source code to which plugins can be incorporated. At this point, we know which functionalities should be included, but we do not know the plugins that must be specified.
2) Another case that may arise is when it is assumed that the developer has already implemented a website with a set of plugins, and from the selection of a plugin already incorporated in the current version, wants to get recommendations with alternatives from different sources which could be of interest to enrich the functionality of the website. In this scenario, the recommendations are obtained based on similar plugins that other developers, with the same profile as the current developer, have used in their websites.
3) Similar to case 2, we assume that the developer has a website implementation. However, the intention is for the site to improve its functionalities by varying the features with plugins that could replace existing plugins. Since a feature can be implemented by several plugins, the application can suffer structural changes in

the number of features and plugins. To achieve this evolution, our approach from a plugin already incorporated in the current version of the site, analyzes its descriptive information (*e.g.*, the set of tags associated with each plugin) and recommends plugins similar to the current one that might interest the developer to enrich and offer new services on the website. For example, if on a website developed in WordPress, the feature *Catalog* has been configured with a specific plugin with the sole purpose of listing the products of the store, the tags associated with this plugin will help to find new plugin alternatives to improve the functionality of the site. In other words, the list of recommendations could offer plugins with additional functions, i.e., in addition to listing the products, they could also allow to compare them with others, view them in different ways and make the payment process quickly.

In Section IV, we describe in detail how RESDEC addresses each of the scenarios described through the use of the proposed algorithms that are commonly used in recommender systems.

IV. THE RESDEC SOLUTION

In this section, we introduce RESDEC (REcommender System that suggest implementation Components from selecteD fEatures). RESDEC recommends component to users by analyzing a repository of rated components, as well as content information from components. RESDEC supplies users with information about which components are more suitable to them in the context of SPL configuration.

RESDEC Requirements: To list the set of valid SPL configurations, computer-assisted mechanisms are needed. The number of valid configurations in an SPL can grow up to 2^n, where n is the number of features. With a large number of features, managing the configuration of a product becomes a difficult task, even more difficult when, for each feature, there is more than one possibility of implementation.

As shown in Fig. 1, when managing an SPL of eCommerce websites based on WordPress, there can be a large number of implementation components (plugins) associated to features, such as payment options or security controls. This relationship between the features and the implementation components is usually too large, so implementing the website with a combination of appropriate components can be a complex activity to solve.

RESDEC uses feature models to describe the set of valid configurations. From these models, we can identify the implementation components associated with each feature and obtain the information developed around them. In our case, we will consider the ratings of the components. The processes for building these feature models are beyond the scope of this paper. Therefore, our approach takes as an input a feature model with components information associated to each feature.

RESDEC Components: Fig. 3 illustrates the components of our proposal. As a first element, we have the *feature model*

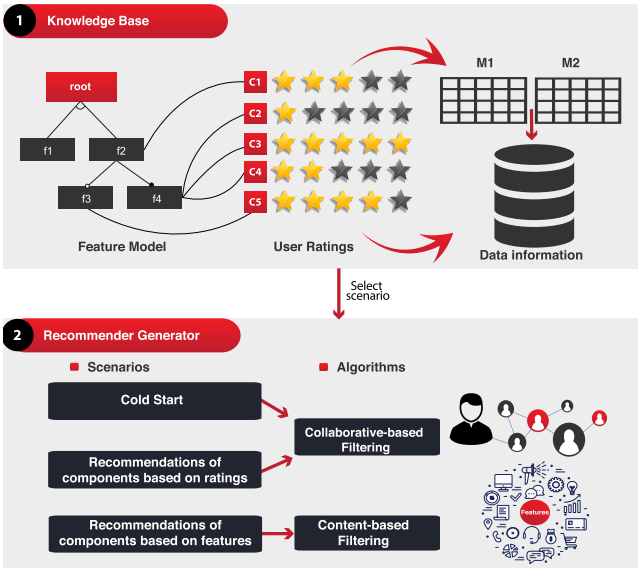

FIGURE 3. RESDEC components.

that describes the variability present in a domain of an SPL. From the feature model, it is possible to select a set of features that are associated to a series of components. Some of these implementation components contain a set of information that is generated by users, such as, the ratings received by these components, bugs reports, number of installations, test cases exceeded, etc.

The information collected from the users allows building the *Knowledge Base* (KB) composed of two matrices, $M1$ and $M2$. Matrix $M1$ relates user information, implementation components and ratings; while the matrix $M2$ relates the information between the implementation components and the associated features (see Section IV-A).

The *Recommender System* contains a set of collaborative-based and content-based filtering recommender algorithms; which are implemented in three different scenarios (see Section IV-B): *(i)* cold start; *(ii)* Recommendations of implementation components based on ratings; and *(iii)* Recommendations of implementation components based on features. Finally, regardless of the scenario and the algorithm, RESDEC recommends a list of implementation components which guides the configuration of a product in an SPL domain. Next, we will describe the information required as an input by RESDEC and the recommender algorithms handled in each identified scenario.

A. KNOWLEDGE BASE

The knowledge base of our proposal contains information about the implementation components which used and rated in previous configurations (see Fig. 3).We will use the term *stakeholder* or *user* to identify the software specialists and the term *implementation components* to represent the items recommended. As we have mentioned, these ratings can represent the quantitative degree of importance and relevance of a component to a specific user. In addition, it contains textual

information (in form of tags) that characterizes each of the implementation components.

The user ratings on the implementation components are represented by an $M_{mxn} \in \mathbb{Z}$ matrix called M1, where the rows determine the m users $(u_1, u_2, ..., u_m)$ and the columns the n implementation components $(p_1, p_2, ..., p_n)$. Each cell $r_{ij} \in \{1...5\}$ represents the rating provided by the user u_i to the implementation component p_j, where 1 represents the worst rating and 5 the best rating. With a similar structure, we represent the features required by each implementation component. In this case we have a matrix $N_{nxk} \in [0, 1]$ called M2, that relates the n implementation components $(p_1, p_2, ..., p_n)$ of the matrix $M1$, with a set $(f_1, f_2, ...f_k)$ of k features. Where b_{ij} determines the presence (1) or not (0) of a feature f_j in the component p_i .

M1

	Plugin 1	Plugin 2	Plugin 3	Plugin 4
Web Developer 1	0	1	3	0
Web Developer 2	5	5	4	0
Web Developer 3	5	3	2	1
Web Developer 4	0	0	0	4
Web Developer 5	1	3	0	5

M2

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
Plugin 1	1	1	0	1	1
Plugin 2	0	1	1	1	1
Plugin 3	0	0	1	1	1
Plugin 4	1	0	1	1	0

FIGURE 4. Example of matrices $M1$ and $M2$.

Fig. 4 shows the representation of the matrices in a scenario with ratings of five stakeholders on four plugins for the configuration of a WordPress-based website. Also, the relationship of each plugin with a set of five features. In this example, it can be seen that it is not necessary for all the plugins to have ratings (see $r_{ij} = 0$ in $M1$, e.g. Web Developer 1 did not rate plugins 1 and 4). Moreover, a plugin does not require all the features (see $b_{ij} = 0$ in $M2$).

B. RECOMMENDATION GENERATOR

In section III, we introduce three common scenarios to guide users configuring components. In this section, we will analyze how RESDEC, depending on the specific needs of the stakeholder or user, addresses these scenarios for the configuration of a product in the domain of an SPL through the selection of optimal implementation components. Next, we present the goal pursued in each scenario and determine which recommender algorithms can be used.

1) COLD START

This scenario occurs when a user starts configuring a product from scratch and need to know what components could be used to implement each feature of a particular configuration. Note that in this case, the user has no experience in the domain of the application and therefore the system does not have information associated with his profile, this is the reason why the system identifies this as a *cold start* and suggests the most commonly used implementation components best rated by the community.

For example, a user wants to set up a website to organize family memories. For this, it is necessary to implement a series of functionalities on the site. In this case, the system makes the recommendations based on the most popular plugins used to implement the features of the site. For another example, suppose the user wants to implement the *video viewer* feature, in this case the approach will filter only those implementation components (plugins) that implement the selected feature. Also, if the user wants to implement the features: *search engine, video and photo viewer, management of albums and social media*, we will use a cold start approach in which we filter the most popular implementation components that suits better the set of selected features.

Goal: Find the set of most suitable implementation components to implement functionalities to the features of new products in an SPL based on popularity level.

Notice that the features will be selected and deselected based on the set of chosen components.

Recommender Process: For the recommender process carried out in this scenario, we implement the *Non-Personalized algorithm* presented in Section II-B.1 by using as input ratings matrix $M1$ to find the most popular implementation components. Table 1 briefly explains the operation of this algorithm.

TABLE 1. Cold start algorithm.

Input: User u_k, Matrix $M1$, recommendation size k
Step 1. Obtain average rating $\bar{r}_{p_j}$ of each component **Step 2.** Select the k best-averaged components $R_{u_k} \leftarrow max_k(\bar{r}_{p_j})$
Output: k most popular components R_{u_k}

2) RECOMMENDATIONS OF IMPLEMENTATION COMPONENTS BASED ON RATINGS

This scenario occurs when the user wants to evolve the implementation components and wants to get recommendations on the better options. Note that in this case, the user already has information associated with his profile, so the recommendations are made based on the user's historical information and its similarity within the community. For example, in custom website developments, it is common to use a series of plugins, which usually have information associated with them (*i.e.*, ratings). This information is what allows the system to identify which plugins similar to those implemented have been used by users in other developments and that could be of interest to the web developer.

Goal: Find implementation components of an SPL based on the ratings and profiles of similar users.

Recommender process: For the recommender process that is carried out in this scenario, we have implemented in RESDEC three recommender collaborative-based filtering

algorithms (CF): *user-user KNN*, *item-item KNN*, and *SVD (Singular Value Decomposition)* presented in Section II-B.1. The proposed collaborative filtering algorithms use as input the matrix $M1$ described in Section IV-A, a user profile (which contains the history of the implementation components installed with the ratings made in the past), and a value k that defines the number of elements to recommend. From this information, the algorithm uses a measure of similarity (in our case, we used Pearson correlation coefficient) from which a recommendation list of k implementation components is obtained.

In Section V, we present a comparative study of the three collaborative filtering algorithms to determine which algorithm is most suitable for recommending implementation components in the evaluated scenarios. Table 2 shows a general outline of the recommendation strategy used in this case.

TABLE 2. General scheme of the collaborative-based filtering algorithms: A) neighborhood-based recommender, and B) model-based recommender.

A	B
Input: User u_k, Matrix $M1$, recommendation size k	
Step 1. Build user history u_k - Set of rated components (p) - Ratings vector (r_{u_k}) **Step 2.** Calculate similarity (S) **Step 3.** Calculate predictions $f_{u_k,p}$ **Step 4.** Select k components with the best prediction $R_{u_k} \leftarrow max_k(f_{u_k,p})$	**Step 1.** Factor matrix $M1$ **Step 2.** Calculate predictions $f_{u_k,p}$ **Step 3.** Select k components with the best prediction $R_{u_k} \leftarrow max_k(f_{u_k,p})$
Output: k components with the best prediction, R_{u_k}	

3) RECOMMENDATIONS OF IMPLEMENTATION COMPONENTS BASED ON FEATURES

This scenario, similar to the previous one, occurs when the user has previous experience in the implementation of components to an SPL configuration environment (*i.e.*, there is already information associated with his profile). The difference with the previous scenario is that here the recommendations are computed considering the feature associated with the implementation components (*i.e.*, content information that allows enriching the recommendations). Specifically, we consider the relationship between features and implementation components associated with the user profile.

For example, supposed that we implemented an eCommerce website in which we incorporated the plugin *payments with credit cards* to implement the *online payments* feature. However, we do not want to replace this functionality but know if there are better implantation alternatives from the existing plugins. In this case, we would use the plugins' tags (*i.e.*, features associated with each component). The idea of using the tags is to allow finding plugins that are different from those already installed but still share certain functionalities. In the aforementioned case, the system could recommend plugins to make payments through *PayPal* or some other means of payment.

Goal: Find components that work as effective implementation solutions to enrich features functionalities in the configuration of an SPL based on component features.

Recommender Process: For the recommender process that is carried out in this scenario, we have implemented in RESDEC the content-based recommender algorithm TF-IDF, described in Section II-B.2. We focus exclusively on the cosine similarity based on TF-IDF due to its simplicity and robustness to work with descriptive information.

In addition, as the information of the implementation components is presented in the form of tags, this algorithm was easy to adapt in configuration environments of an SPL allowing to find the relevance between components according to the content. However, in the literature, there are other algorithmic proposals used in content-based recommender systems, which we have considered as future work.

This algorithm uses as input the matrices $M1$ and $M2$ described in Section IV-A (which contains the history of the features and implementation components that the user has used in the past), and a value k that defines the number of elements to recommend. Thus, from matrix $M1$ we obtain the profile of the user u_k, and from matrix $M2$ we obtain the features associated with each component. Given this process, the system calculates the w weighting for each feature f and then establishes the similarity between the features f of the selected p component and the features of all the components associated with the user profile.

Finally, a recommendation list of k implementation components p is returned to user u_k. Table 3 shows the general scheme for making recommendations in this scenario using content-based algorithms.

TABLE 3. General scheme of the contend-based algorithm.

Input: User u_k, matrix of ratings $M1$, matrix of features $M2$ recommendation size k
Step 1. Build user history u_k - set of rated components (p) - Ratings vector (r_{u_k}) **Step 2.** Obtain from $M2$ the features associated with the components (p) **Step 3.** Calculate the (w) weighting for each (f) feature. **Step 4.** Calculate similarity between features f (S) **Step 5.** Calculate the predictions $f_{u_k,p}$ **Step 6.** Select the k components with the best prediction $R_{u_k} \leftarrow max_k(f_{u_k,p})$
Output: k components with the best prediction, R_{u_k}

V. EVALUATING RESDEC

Developers and configurators who creates Wordpress sites face the challenge of choosing the plugins that best suits their needs. In this context, RESDEC aims to assist them in the decision making process.

Figure 5 shows the process we performed to evaluate our approach. First, we describe how we extracted the data describing the available plugins at Wordpress and how we build an example feature model describing a product line of eCommerce wordpress sites. Later, we go through the different experiments we developed to test different scenarios where we envisioned RESDEC to be used.

A. EXPERIMENTATION DATA

1) WORDPRESS DATA

To obtain the data for these experiments we have created a selenium based crawler[2] that extract the different matrices used in this paper. First, we extract the list of plugins from

[2]Selenium WebSite: https://www.seleniumhq.org/

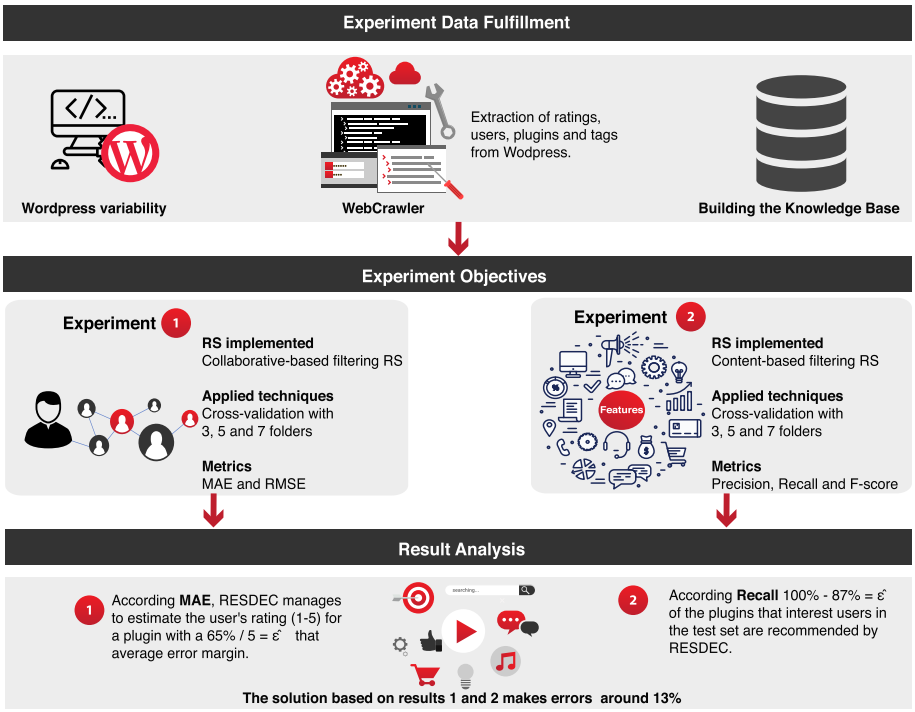

FIGURE 5. Evaluation steps performed to evaluate RESDEC.

Wordpress,[3] then, through the different plugins obtaining its number of stars, downloads, version, last update date, the wordPress version, the required PHP version, and associated tags. Finally, we obtain the list of users that reviewed the plugin and scores. This information is then stored in a Json file which is later exploited to generate the required inputs for RESDEC. Both the crawler and the used data can be found at RESDEC website.[4]

2) WORDPRESS FEATURE MODEL

To build the feature model based on a software products line of eCommerce sites on Wordpress, we have followed a systematic process, ranging from searching for eCommerce websites developed on this platform, the plugins that have been implemented, identifying features and assigning the plugins to each of them. The process consisted of five phases:

- *Searching of eCommerce Websites:* As a first step, a search of the websites that have been developed on Wordpress was made, specifically those of eCommerce type, discarding the sites that do not correspond to this category. For this process, we looked for success cases of companies whose websites have been designed on this platform. As the first searching criteria, we considered the showcase published on the official Wordpress site[5] where we obtained a list of seven websites. Then, we searched throughout the ranking of websites of the last four years (sites developed on Wordpress that are currently working). As a final result of this search, we obtained a total of 28 websites.
- *Implemented Plugins Identification:* Once the list of websites was obtained, the next step was to identify which Wordpress plugins are implemented to each one of them. For this process, we used the Wptheme detector tools[6] and Wordpress Plugin Checker[7] where we obtained a list of plugins for each site. In some cases, we checked the source code of the site to identify plugins that were not detected by the aforementioned tools. From the list we obtained, we reviewed those that are still valid for their use and discarded those that are no longer supported.
- *Construction of Features From Plugins:* With the list of plugins we obtained in the previous phase, we were able to identify what the possible features of our model would be. This process consisted in classifying each plugin by category. For example, those used to implement the product catalogue, shopping cart, online payments, security controls, among others.
- *Feature Model Development:* We started to build the feature model once the plugins were classified by category. Each category became a parent feature of the model, however, in some cases we had to unify categories to form a single parent feature and in other cases, a category became a sub-feature of a parent feature.

[3]Wordpress plugins: https://wordpress.org/plugins/browse/popular/
[4]Selenium WebSite: http://resdec.com/
[5]Wordpress showcase: https://wordpress.org/showcase/
[6]Wptheme site: https://www.wpthemedetector.com/
[7]Wordpress Plugin Checker site:http://wppluginchecker.earthpeople.se/

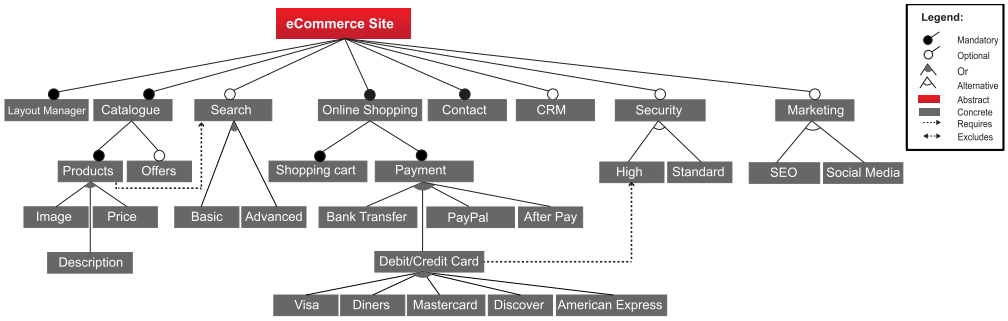

FIGURE 6. A feature model describing an eCommerce-based website on Wordpress.

Fig. 6 shows the feature model built from the results of the steps described above. The model shows the relationship between each of the features, as noticed, some relationships are mandatory and others optional; it also shows some restrictions that are considered during the configuration of an eCommerce website.

- *Plugin mapping to feature model:* With the feature model built, the next step was to map the plugins, which were previously classified by category. For this process, we associate each feature to the plugins that could be used to implement it. In this case, the same plugin could implement more than one feature, so this association is allowed.

B. EXPERIMENT 1: VALIDATION OF RECOMMENDATIONS BASED ON RATINGS

In this first experiment, we want to understand if the recommendations retrieves were accurate enough when using the collaborative filtering approaches in RESDEC. To perform the evaluation of the algorithms, we conducted an offline evaluation [37]. We simulated the online process where the system makes predictions by building the recommender systems using a part of the knowledge-base (training set). Then, we evaluate its performance with the rest of the data (test set).

Particularly, we used a cross-validation approach to prevent bias caused by the selection of the training and test sets [38], [39]. Cross-validation consists on generating different sets of data (one set per fold which is a parameter of the validation procedure) later used as training and test inputs. The procedure iteratively selects one fold as test and the rest as training to evaluate an algorithm and provides a mean of the metrics obtained for each iteration. In our case, we performed cross-validation with three, five and seven folds, repeated five times and collected the mean accuracy (see metrics in section V-B.1). Beyond the type of data we were analyzing, we wanted to know the robustness of the algorithms regarding the training/test distribution, which in a 10-fold CV is 90/10%.

Using 3-, 5- and 7-fold CV we obtain results for training/test data distributions of 66/33%, 80/20% and approximately 85/15% which gives us an idea of the models quality similar to the 10-fold cross-validation but it brings the idea of how stable the algorithms are regarding sudden changes in the data distribution.

All parameters of the algorithms used in this experiment were set to defaults in the Scikit-surprise library (see Appendix VIII).

1) METRICS

Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are two of the most common metrics (see Eq. 11 and 12, respectively) used to measure accuracy for continuous variables [40]. In this experiment, collaborative-based recommender systems provided an affinity score for every pair of plugin and user. The affinity score is a continuous variable later used for getting recommendations with the best fit.

The algorithms used in this experiment started from the experience of users with similar profiles and the ratings they made on a set of plugins. As mentioned in Section II-B.1 these algorithms predict the ratings of users and from these predictions, select the best plugins as recommendations. In this context, we used MAE and RMSE [37] to measure the goodness of fit when predicting ratings and determine how close the score calculated by RESDEC was to the actual score that a user gave to a plugin.

$$MAE = \frac{1}{\Upsilon} \sum_{(u,i)\in\Upsilon} \left|\hat{r}_{(u,i)} - r_{(u,i)}\right| \tag{11}$$

$$RMSE = \sqrt{\frac{1}{\Upsilon} \sum_{(u,i)\in\Upsilon} (\hat{r}_{(u,i)} - r_{(u,i)})^2} \tag{12}$$

For each pair of ratings $\hat{r}_{(u,i)}$ (prediction by RESDEC) and $r_{(u,i)}$ (real rating not used to train RESDEC), the accuracy metrics are calculated as follows: MAE is computed as the mean sum of the absolute errors from predictions of a test set Υ (never used to build the recommender) of ordered pairs user-plugin (u, i). RMSE follows a similar equation but in this case, it calculates the second sample moment of the differences between predicted values and observed values. MAE and RMSE are never negative, but the closer they are to zero, the better performance is provided.

TABLE 4. Evaluation results applying collaborative-based recommender systems under different training sizes. Numbers in bold indicate the best results.

User-User kNN		
Training percentage	**MAE**	**RMSE**
85% Training (7 folders)	0.72	1.40
80% Training (5 folders)	0.72	1.40
66% Training (3 folders)	0.73	1.39
Item-Item kNN		
Training percentage	**MAE**	**RMSE**
85% Training (7 folders)	**0.65**	**1.37**
80% Training (5 folders)	**0.65**	**1.37**
66% Training (3 folders)	**0.67**	**1.38**
SVD		
Training percentage	**MAE**	**RMSE**
85% Training (7 folders)	1.35	1.70
80% Training (5 folders)	1.46	1.81
66% Training (3 folders)	1.83	2.17

2) RESULTS

Table 4 shows the mean results of the evaluations performed as a function of the percentage of the knowledge base reserved for training (66%, 80% and 85% in cross validation with three, five and seven folds, respectively). Figures 7 and 8 show graphically the trend shown in the Table 4.

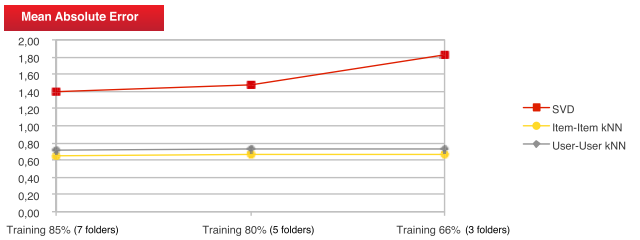

FIGURE 7. MAE Evaluation.

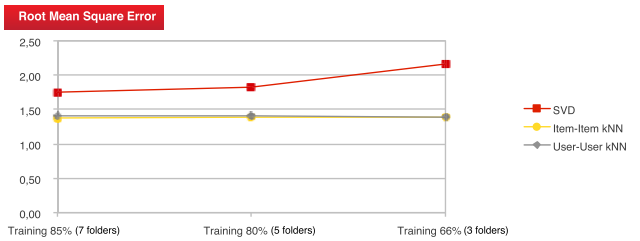

FIGURE 8. RMSE Evaluation.

We observe that as the training base decreases (and therefore the test base increases), errors increase. In the case of SVD, the dependence on the size of the training set is higher. It can also be observed that the results of the *Item-Item kNN* algorithm has a clear advantage over the *SVD* algorithm, since *MAE* and *RMSE* are significantly lower. However, the difference with *User-User kNN* is not so significant. To clarify this point, we performed a statistical validation with the results obtained for each fold and technique in three-, five- and seven-fold cross–validation (15 partial results per technique-i.e., 45 in total).

To validate the performance differences between multiple algorithms of machine learning, it would be common to use an ANOVA(Analysis of Variance) statistical test. The problem is that this type of parametric statistical tests has strong assumptions regarding the data (normality, homoscedasticity, etc.). Instead, it is possible to use nonparametric variants such as the Friedman test. We applied the Aligned Friedman test to study the statistical significance of the differences among the average rankings obtained by each technique and a Holm's post-hoc procedure that allowed us to compare the results of each technique by pairs, controlling the family-wise error by means of the StatService tool [41]. A more in-depth description of the statistical validation performed can be found in [42].

TABLE 5. Mean rankings obtained after comparing the MAE results.

Recommender	Mean Ranking
Item-Item kNN	11,333
User-User kNN	19,667
SVD	38,000

TABLE 6. Mean rankings obtained after comparing the RMSE results.

Recommender	Mean Ranking
Item-Item kNN	13,400
User-User kNN	17,600
SVD	38,000

The aforementioned statistical procedure is based on the generation of mean rankings for each technique. To this end, we generated a ranking for each technique result. The ranking varies from 1 (the best) to 45 (the worst ranking and also, the number of total partial results registered). Table 5 and 6 show the mean rankings obtained for every algorithm regarding MAE and RMSE as accuracy metric, respectively.

We set as null hypothesis for the Aligned Friedman test that *the differences between the mean rankings are not significant*. The p-value for Aligned Friedman test (see [42] for further details) was lower than 0.007 regardless using MAE or RMSE. The test rejected the null hypothesis with a significance level of $\alpha = 0.05$ for both MAE- and RMSE-derived rankings.

The next step was a pairwise comparison to detect if there was significant differences between the best recommender (Item-Item kNN) and each of the other possible options. This was done by relying on a Holm post-hoc test constraining α on each run. We can see on Table 7 that there were significant differences between Item-Item kNN and SVD but not regarding User-User kNN since the p-values could not reject the null hypothesis (the p-value obtained was not equal or lower than the corresponding Holm's α).

Considering the previous analysis, we can observe that kNN collaborative filtering algorithms presented better results in RESDEC for the WordPress dataset scenario. Taking into account that RESDEC obtained a MAE of 0.65 and that we had 5 possible scores we obtained a relative error level of 13% (0.65/5). RMSE reported a higher error

TABLE 7. P-values of the statistical tests of Aligned Friedman between the control recommender (Item-Item kNN) and the rest according to the rankings obtained by MAE and RMSE.

MAE pairwise comparison test results		
Recommender	**p-value**	**Holm's α**
SVD	0,000	0,025
User-User KNN	0,082	0,050
RMSE pairwise comparison tests results		
SVD	0,000	0,025
User-User KNN	0,381	0,050

since it follows a quadratic scoring rule that increase the average magnitude of the error. Higher RMSE than MAE involves eventual larger errors in the distribution. In other words, the MAE-derived relative error can be around 13%, RMSE shows that there can be cases where the error is much higher.

C. EXPERIMENT 2: VALIDATION OF RECOMMENDATIONS BASED ON FEATURES

In this second experiment, we wanted to understand how well RESDEC retrieves configurations when using content-based filtering. Again, to perform an evaluation of the algorithm used, we conducted an offline evaluation simulating the online process by a three-, five- and seven-fold cross-validation approach repeated five times to reduce bias due to random selection of test sets.

1) METRICS

To evaluate the quality of content-based algorithms, we used *Precision, Recall & F-score* since they are metrics commonly used in the literature (see [6], [43]) to describe a recommender system accuracy regardless any rating given by users. Equations 13, 14 and 15 show their calculation.

$$Precision(L) = \frac{1}{u}\sum_{u\epsilon U}\frac{|L(u)\cap T(u)|}{|L(u)|} \tag{13}$$

$$Recall(L) = \frac{1}{u}\sum_{u\epsilon U}\frac{|L(u)\cap T(u)|}{|T(u)|} \tag{14}$$

$$F\text{-}score(L) = \frac{2 * Recall(L) * Precision(L)}{Recall(L) + Precision(L)} \tag{15}$$

Precision for a user u is obtained by dividing the intersection set of the recommended $L(u)$ and the relevant $T(u)$ plugins the user u, among the size of the total recommended set. *Recall*, divides the same intersection by the total of relevant plugins for that user. In both cases, the relevant plugins refer to those that the user has rated in the test set, while the recommended ones correspond to those the system has provided to a particular user after training the recommendation system (but not including interactions from test sets). Precision and Recall are in the interval [0, 1]. A score is considered perfect when it is 1, while 0 is considered the worst possible value. An important issue regarding Precision and Recall is that, in general, Recall increases as the number of recommended items do. On the contrary, Precision decreases when the system provides a higher number of recommendations. F-score combines both measures to mitigate both unwanted effects.

2) RESULTS

In this evaluation, we used the metrics *Precision*, *Recall* and *F-score* for recommendation of 5 and 10 plugins (usually the numbers of recommendations in the literature) of the scenario presented in IV-B.3. The results are shown in Table 8.

TABLE 8. Mean number of ratings by user and 3-fold (66% training set) cross-validation results for our content-based recommender.

Metric	Value
Precision@5	0.44
Precision@10	0.23
Recall@5	0.82
Recall@10	0.87
F-score@5	0.57
F-score@10	0.36
Mean number of ratings by user	2.66

In the content-based scenario, RESDEC worked better with smaller training databases (i.e., with a lower number of folds in the cross-validation). This fact is due to a specific knowledge base (in our case, Wordpress dataset) and dependence of *Recall* and *Precision* on the mean number of ratings by user. In that case, the intersection between recommended and relevant items is low while the number of recommended items will be constant, which makes Precision decrease. The same way, regarding Recall, its value will too often depend on a one-item-only relevant set for a user which may easily increase the number of users without any positive recommendation. In few words, when we have users with few interactions in test, *Precision* and *Recall* may mislead conclusions when recommending a fixed (usually 5 or 10) number of plugins. In our Wordpress database, cross-validation with more than 3 folds provided (on average) test sets with less than 2 plugins rated by a user (the mean number of ratings when using three, five and seven folds was 2.66, 1.6 and 1.21, respectively). This fact made us discourage the inclusion of results of 5- and 7-fold cross-validation for analysis.

From our results on 3-fold cross-validation, 82% and 87% of most-suitable plugins recommended by RESDEC were (on average) really interesting for users according to Recall at 5 and 10, respectively. We observed that precision was lower than Recall. Again, a low mean number of interactions on test could be the reason. Precision depends more on the number of relevant items since divisor in its formulation is constant. As was expected, with a higher number of recommended items, Precision decreased whilst Recall lightly increased. Thus, F-score gave a better idea of what was really happening on both previous measures and concluded that a lower number of items would better fit our dataset.

D. THREATS TO VALIDITY

Although the experiments presented in this document provide evidence that the proposed solution is valid, there are some

TABLE 9. RESDEC vs. others proposals.

Papers	Features	Implementation Components	Collaborative	Content	Algorithms	Tool
Galindo et al. [12]	✓				3	✓
Al-Hajjaji et al. [44]	✓				3	✓
Mazo et al. [45]					6	✓
Martinez et al. [46]			✓		1	✓
Pereira et al. [10]	✓		✓		3	✓
Pereira et al. [14]	✓		✓		4	
Pereira et al. [13]	✓		✓		4	✓
Pereira et al. [15]	✓		✓		1	
RESDEC	✓	✓	✓	✓	5	✓

assumptions we have made that may affect its validity. In this section, we discuss the different threats to validity that affect evaluation.

- ***External Validity:*** The data used for the experiments presented in this paper is based on a realistic feature model based on a product line of eCommerce websites in Wordpress. However, as it was developed following a manual design process, it could have errors and could not represent all the necessary features to configure a website in this domain. To deal with this threat, we perform a systematic search based on cases of website successes created in Wordpress. Then, we filter those categorized as eCommerce and analyze their structure to determine the implemented features. Finally, we made sure that the features of the model represent all the valid configurations for an eCommerce website created in Wordpress.
  Regarding the implementation components, represented by plugins, which were associated with the features of the model, we did not consider versions nor the validity of them within the platform. To address this threat, we identify which plugins have been installed on the sites that were taken for the study. In some cases, tools were used to detect plugins and in others the source code was revised. Finally, we map each of the plugins with the features to guarantee the validity of the feature model.
- ***Internal Validity:*** A threat to internal validity is the selection of recommender algorithms used for the evaluation. In this work, we have included recommendation algorithms based on collaborative filtering and content filtering. However, we have only used one technique of filtering by content and has not been compared with others, as we did with the techniques based in collaborative filtering. To address this threat and support the process of selecting components in the configuration of products in an SPL, as part of our future work we plan to expand our experiments with other types of algorithms based on content filtering.
  Another possible threat is the evaluation technique that we have used in this work. Although it is true that cross-validation was more precise to evaluate the recommendations obtained from RESDEC, we left aside other techniques that could be considered. Thus, there could be other validations to demonstrate which of the algorithms provide better results, precision and performance in the proposed scenarios.
- ***Sensitivity Parameters:*** One of the important aspects that we have not included in this work is the analysis of the optimal values that should be introduced in the algorithmic models when making the recommendations of the implementation components.
  In this context, for future work we hope to perform a parametric sensitivity analysis as well as a detailed study of the parameters that are introduced in each of the proposed algorithms. The objective that we seek with this analysis is to know how each parameter influences in the final results. For this purpose, we will carry out a simulation by varying values in each parameter based on a central value and analyze the effect that these changes cause in the recommendations given by RESDEC.

VI. RELATED WORK

In this section we compare our proposal with existing literature. In Table 9 we summarize the main characteristics of RESDEC and compare them with other proposals

The column *Papers* of Table 9 shows some works directly related to our proposal. First, we will review in the literature proposals based on the selection of configurations, then, address works related to recommender systems applied in an SPL. All proposals presented use feature models.

The *Features* and *Implementation Components* columns in Table 9 indicate whether the cited works employ features and implementation components in their proposals. As it is observed, only the proposals presented by Galindo *et al.* [12], Al-Hajjaji *et al.* [44] and Pereira *et al.* [10], [13]–[15] make use of features. On the other hand, none of the proposals use implementation components.

In the *Collaborative* and *Content* columns of Table 9 we refer to the recommendation algorithms we have used in RESDEC, with which we determine whether the proposals presented use some of them in the configuration selection process; while the *Algorithms* column indicates the number of algorithms implemented.

To learn more about the proposals presented in this section, we briefly analyze each one of them:

In the work presented by Galindo *et al.* [12] the authors present a solution to prioritize configurations for testing based on value attributes, in this case the cost of testing. The proposal incorporates a prototype tool that processes a set of configuration rules for an SPL, given by the developer, through the use of cost and value functions. The proposal

does not use any recommendation technique nor does it have a dataset with historical data, since the information it processes is generated manually and is not constantly updated.

Al-Hajjaji *et al.* [44] presents a proposal to prioritize configurations based on the similarity between one or more configurations. The hypothesis handled by the proposal supposes that, if a configuration presents some failure, it is probable that the similar ones also have it, having as a result more possibilities to quickly detect errors.

The techniques used to prioritize configurations according to similarity are not based on algorithms commonly used in recommender systems. On the other hand, the data used in the solution is derived from a model that is previously generated manually; that is to say, like the previous proposal, there is no mechanism that generates and automatically updates the information that is processed.

The work of Mazo *et al.* [45] presents a proposal that through a collection of heuristics and using programming with restrictions, seeks to improve the process of configuring a product to reduce the number of steps and the time to prove the validity of the product line. This work is the first one that introduces short-term recommendation techniques since it uses six algorithms to solve each of the presented heuristics. However, it is not determined whether the information shown to the user for the configuration of a product is actually known or useful, or based on the experience of other users in similar configuration processes, for example, which questions whether the final configuration of the product will satisfy the user's requirements.

In the proposal of Martinez *et al.* [46] the authors propose the use of an interactive genetic algorithm for the selection of a relevant set of configurations for users. The proposal uses a dataset with information of configurations valued by users that is exploited by data mining techniques. Although it is true that the proposal contains all the elements that a recommendation system needs (users, items and ratings), the way how the algorithm operates in its entirety is not presented in detail. By the use of the genetic algorithm and the information that is used we could say that we are employing recommendation techniques based on collaborative filtering.

In the work of Pereira *et al.* [10] the authors present the first proposal in which a well-structured recommender system is used to configure products in an SPL. The proposed solution involves the user throughout the product configuration stage guiding the selection of features that best suit their requirements. However, the information presented to users for the configuration of the product does not come directly from criteria given by users in the past; that is, the features shown for the configuration do not have an indicator that determines if those features have been evaluated positively or negatively by the users, which makes it difficult to determine if a selected feature has been implemented successfully in past configurations.

Following the same approach as in the previous proposal, Pereira *et al.* [13] show an extension of the work presented in [9] in which it is present a tool that improves the visualization aspects for the configuration of products in an SPL by means of a recommender system based on non-functional properties (NFPs) of the features. The objective of the proposal is to ensure the consistency of the configured products and reduce the effort of those responsible for the configuration thereof.

The same authors in the work [14] present a solution that uses a recommender system to predict the features during the configuration of products using contextual information of the users, specifically, the requirements that the users define for a product. In the first proposal the user defines the requirements of the product to be configured, then the system performs a search to include historical data based on the specified requirements. Later, it creates a list of features that help the user to identify those features relevant for the configuration of the product and finally, the system checks the integrity of the configuration verifying if there are some undefined features. In case the system finds a partial configuration, it predicts the features for said configuration, complements it and shows them to the user, thus improving the general quality of the recommendation.

Finally, in the work [15] a new proposal for the configuration of products is presented based on a recommender system that uses contextual information of the users. In this case, new analysis dimensions are introduced, which go beyond the two typical dimensions, users and items, which are usually used by a recommender system. For this purpose, the authors introduce the technique of collaborative filtering Tensor Factorization (TF) [47] to automatically prioritize the features and auto-configure the SPL at execution time according to the contextual information that it originates around users and improve performance in the process of configuring a product.

All the proposals of Pereira *et al.* which have been mentioned above, employ a well-structured recommender system whose main elements are users, items and ratings. In addition, they make use of a historical knowledge base that is constantly updated. However, it can be perceived that the obtained ratings do not come from opinions that common users would give to a configuration of a product (such as errors during operation, poor design, among others). On the contrary, the information that is used is probably collected from the opinions of expert users, since common users could not accurately evaluate technical aspects related to the configuration of a product. On the other hand, the proposals only use collaborative-based recommender systems and focus exclusively on the configuration of products, leaving aside the components that implement the features during the configuration of an SPL.

Comparing the works described above with RESDEC, based on the characteristics of columns [4-7] of table 9, we can say that one of the main advantages of our proposal is the use of a knowledge base that feeds of information that comes from the experience that ordinary users have had when using products in an SPL domain. Specifically, in our proposal this information is represented by the ratings that users have made about the components used to implement

features in the configuration of a product. We believe that it is more feasible and real, to obtain user evaluations about the implementation components, rather than the features of the product; since as mentioned before, a common user would not have the experience to evaluate technical aspects related to a product.

On the other hand, we are the first to introduce the concept of Component-based Recommender Systems in the configuration of an SPL through the use of collaborative-based filtering and content-based filtering techniques.

The column *"Tool"* in Table 9 indicates whether the proposals include some prototype tool as part of contributions. All the proposals, except those presented by Pereira *et al.* [14], [15], propose a tool for configuring products in an SPL. However, RESDEC is the only tool that has been implemented using a well-defined scheme of recommender systems; and besides, it is the only one that incorporates a set of algorithms that are executed in three different scenarios that validate the platform's capacity in terms of operationalization and scalability.

Finally, all proposals include an evaluation with information obtained from various business sectors.

VII. FUTURE WORK

A. USE OF IMPLICIT INFORMATION

In the proposal presented in this paper, we have only considered recommender techniques that take into account explicit information of the users (*i.e.*, ratings). As future work, we aim to include implicit information from events defined indirectly by the user, such as, number of clicks, number of views, etc. For example, in addition to including user ratings on the plugins, we could also include implicit information as the number of downloads, number of views and versions of the plugins. Information that would allow the recommender system to work in a more personalized way. In the literature, there are several recommender algorithms that use implicit information to make the recommendations [48], [49]. Our future aim is to adapt this algorithms to enrich the recommendations of implementation components in the domain of SPL configuration.

B. USE OF CONTEXTUAL INFORMATION

We aim at recommending components based on contextual information derived from the users, features and configurations by adapting the algorithms proposed in Pereira *et al.* [10] and which can be processed according to the RESDEC components presented in figure 3.

The objective that we seek with the implementation of this scenario is to guide the user during the process of configuring a product. This way, as the user progresses in the configuration of a product, RESDEC is able to automatically suggest which feature can be selected to complement the partial configuration based on associated descriptive information to the features, users and configurations.

To face the handling of contextual information and offer a personalized product to the user of better quality, we intend to extend the benefits offered by the techniques of Factoring Matrix by introducing the technique of collaborative filtering Tensor Factorization (TF) [15] that allows an integration of contextual data that does not focus only on information from matrices of user and items. In our case, this technique will allow us to explore beyond the contextual information of the components of implementation of the features, and will facilitate us to involve other dimensions of study such as, for example, contextual information of the features and configurations not considered in this work.

An approximation to this type of recommendation could be the following, suppose that the user who set up a website for *tourism promotion* also set up a website for *travel*; when new users set up a *tourism promotion* website, it is likely for the system to recommend the configuration of the website for *travel*. Note that in this case we make use of contextual information that is developed around a valid configuration of a product.

C. RECOMMENDATION OF CONFIGURATIONS FOR TESTING

The objective is for RESDEC to be able to recommend configurations more susceptible to errors and therefore could be candidates for testing. The aim is to provide to the person in charge of supervising the quality of the SPL an automated mechanism that allows him to select the configurations more error-prone. For this, we are based the hypothesis that, the configurations with the lowest rating by the users are those which tend to contain more errors. For example, if we have designed a mobile application for *tourist promotion*, the system should be able to recommend which configurations of mobile devices testing should be performed. For this purpose, mobile devices in which similar applications have had unfavorable ratings will be recommended for testing. To make this type of recommendations, we will use the RESDEC recommendation elements and algorithms shown in Figure 3.

VIII. CONCLUSION

Managing the SPL configuration process is a complex task for the software developer, even more complex when there is more than one possibility of implementation for a feature. In this scenario of constantly evolving products, the selection of implementation components becomes a difficult task. As we mentioned in this paper, a common example is found in the web development industry during the selection of plugins among a wide variety of options to choose from; in such a case, selecting empirically a plugin could not provide the expected results and consequently provide a bad experience for the users.

In this work, we have presented RESDEC as a proposal to support the user to face the difficulties that occur when selecting implementation components to configure an SPL. To this end, we have identified three possible scenarios in which we can make use of explicit information from the users and the implementation components. The first scenario

called *Cold start* that recommends components when there is no information associated with the user profile, that is, when the user has not had experience and for the first time is going to configure a product. The second scenario called *Recommendations of implementation components based on ratings*, which, based on the components linked to the user profile, recommends components that other users have used in past configurations. Finally, the scenario *Recommendations of implementation components based on features*, which recommends implementation components based on the features of the components associated with the user profile, that is, in the descriptive information of the components.

The modeling of the problem for the implementation components selection using collaborative-based and content-based recommender systems algorithms and the design of a prototype tool for RESDEC are the new contributions of this research. The results obtained in the evaluation carried out using a WordPress dataset show that RESDEC is capable of making recommendations on implementation components with an error lower than 13%.

As future work, we aim at addressing the following issues:

- Explore new recommender algorithms.
- Conduct user empirical study to identify the usability of RESDEC.
- Explore other domains by using a knowledge base different from WordPress.
- Extend RESDEC Tool to be able to make recommendations in real time.

This work present preliminary results to support the configuration of implementation components in the domain of SPL. Moreover, this proposal could be also applied to other environments that face similar problems, such as, the selection of deployment environments for mobile applications.

MATERIAL

The RESDEC source code can be downloaded from the project repository **https://github.com/RESDEC**. The tool prototype using information extracted from WordPress is available on the RESDEC website **http://resdec.com/**.

APPENDIX: RESDEC TOOL

RESDEC Tool (see Figure 9) is a prototype recommender tool that provides real-time assistance in the selection of implementation components to configure the features of a product belonging to a domain of an SPL. It is designed to provide support and help stakeholders with or without information associated to the user profile.

The recommender process takes place in two stages: 1) For stakeholders that need to configure a new product, RESDEC provides recommendations based on the trend of the most ranked and used implementation components within the community, 2) For stakeholders with previous experience, RESDEC provides a list of recommendations on implementation components based, on the one hand, user profiles that have used similar components and; on the other hand, based on the features associated to the implementation components.

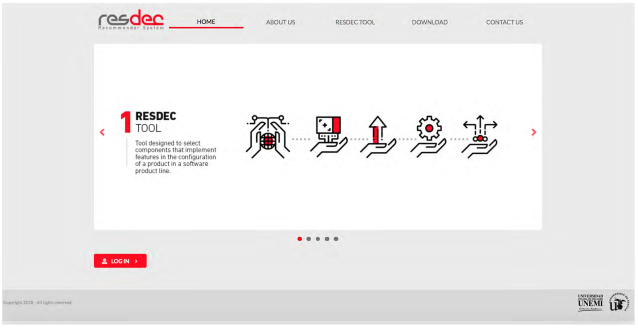

FIGURE 9. RESDEC recommender system.

For example, if the stakeholder wants to set up a website in WordPress, RESDEC offers a list of plugin recommendations based on the experience of users who have set up similar websites in the past. These recommendations are carried out through personalized searches through which the stakeholder can find suggestions according to the popularity of the plugins, or based on the tags of the plugins used by the community and thus produce the desired result.

RESDEC uses as input data information from several data sources, such as CSV's, database, among others, which are fed by information that is developed around the implementation components of a Feature Model. As mentioned in section IV, the construction of these models is beyond the scope of this proposal.

In this work we only use CSV files as a data source. The data collected by these files correspond to users, ratings of the implementation components and the features associated to these components.

The recommendations are calculated by using information from several mandatory fields provided by users, such as the implementation components used in past developments, the features of the implementation components and the number of recommendations that the system returns. With the information given by the stakeholder, RESDEC runs recommender algorithms that use similarity functions to determine the similarity between the implementation components used by the stakeholder and those used by the community. Algorithms such as the similarity functions implemented in RESDEC can be extended according to the stakeholder's needs.

As a final result, RESDEC shows the stakeholder a list of recommendations that allows him to save time in the selection of implementation components and at the same time provides useful information not considered or not known as possible to fit the goals of the product that is desired to be configured or extended. Along with the list of recommendations, RESDEC shows a tab called "You might also be interested" that shows other components that could be of the interest to the stakeholder.

To sum up, RESDEC offers the following advantages: (1) Information about users, implementation components and ratings, used to build recommendations from feature models.

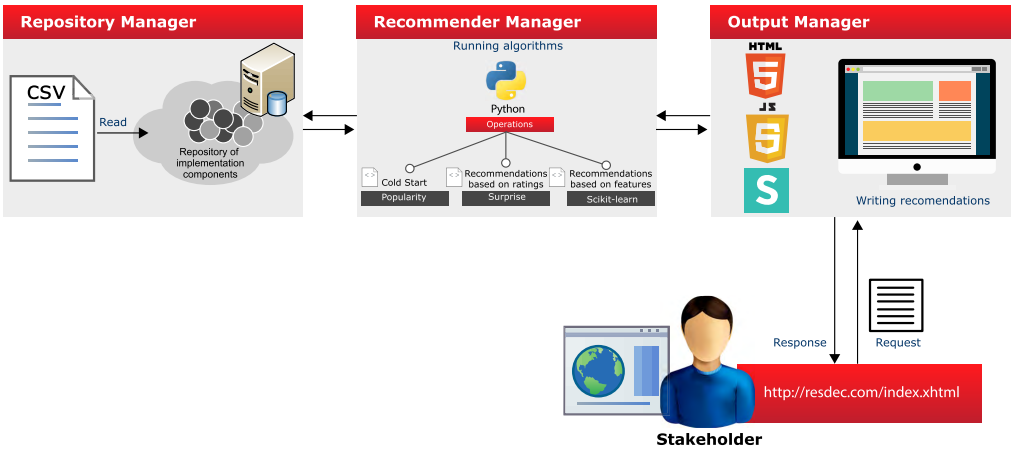

FIGURE 10. RESDEC architecture.

(2) It is easy to adapt to any application environment of an SPL (3) It provides a set of recommendation algorithms that can be extended and adapted in any of the three scenarios presented in this paper. (4) It provides several data sources, being these CSV's, database, among others. (5) It offers a search history of the last implementation components that the stakeholder used to produce the recommendations.

A. RESDEC ARCHITECTURE

RESDEC Tool has 3 components (see Figure 10): a repository manager, a recommender manager and an output manager.

The *repository manager* responds to the requests of the stakeholders and structures the matrices $M1$ and $M2$ of the *Knowledge base* presented in section IV-A through CSV's.

The *recommender manager* is in charge of processing the recommendations based on the three scenarios presented in section IV-B. It is developed in *Python* with a package of libraries which contain the algorithms that the recommender manager runs according to the scenario selected by the stakeholder.

For the *Cold Start* scenario presented in section IV-B.1 RESDEC uses a classical popularity algorithm. While for the algorithms which run in the scenario *Recommendation of implementation components based on ratings* presented in section IV-B.2, employs the Scikit-surprise library[8]; and for the *Recommendation of implementation components based on features* scenario presented in section IV-B.3, it uses the Scikit-learn library.[9]

The recommender manager was designed to be scalable over time, that is, it offers the possibility of extending the benefits of RESDEC by allowing the adaptation of new similarity metrics and new recommender algorithms for each of the scenarios to which a stakeholder could face.

The *output manager* interacts directly with the stakeholder using the repository manager and the recommender manager to generate the list of suggestions for the implementation components. It is designed in HTML5 and JavaScript, supported by the Semantic UI framework[10] used for the design of the interfaces. The interaction between the stakeholder and RESDEC is done through a web browser.

In general, the *output manager* is responsible for receiving the requests of the stakeholders and informing the *recommender manager* of the requirements so the appropriate algorithm is run with the information that the *repository manager* responds, it finally displays the generated recommendations on the screen.

B. RESDEC WEB

To make our work accessible to the community, we present a RESDEC web application that eases the generation of recommendations to stakeholders that require guided assistance in the selection of plugins to configure a line of products based on WordPress websites. The application is available at http://www.resdec.com.

ACKNOWLEDGMENT

The authors would like to thank Mathieu Acher, Associate Professor at Université of Rennes 1; and Juliana Alves Pereira, Post-doctoral researcher at INRIA both members of the DiverSE team (INRIA/IRISA) for their help reviewing this paper that clearly helped us to improve it.

REFERENCES

[1] M. A. Babar, L. Chen, and F. Shull, "Managing variability in software product lines," *IEEE Softw.*, vol. 27, no. 3, pp. 89–91, May/Jun. 2010.
[2] M. Lettner, J. Rodas, J. A. Galindo, and D. Benavides, "Automated analysis of two-layered feature models with feature attributes," *J. Comput. Lang.*, vol. 51, pp. 154–172, Apr. 2019.
[3] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," DTIC, Softw. Eng. Int., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-90-TR-021, 1990.

[8]Surprise website: http://surprise.readthedocs.io/en/stable/
[9]Scikit-learn website: http://scikit-learn.org/stable/index.html
[10]Semantic website: https://semantic-ui.com/

[4] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Inf. Syst.*, vol. 35, no. 6, pp. 615–636, 2010.
[5] J. A. Galindo, D. Benavides, P. Trinidad, and A.-M. Gutiérrez-Fernández, and A. Ruiz-Cortés, "Automated analysis of feature models: Quo vadis?" *Computing*, vol. 101, no. 5, pp. 387–433, 2019.
[6] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowl.-Based Syst.*, vol. 46, pp. 109–132, Jul. 2013.
[7] Y. Gonzalez-Fernandez, S. Hamidi, S. Chen, and S. Liaskos, "Efficient elicitation of software configurations using crowd preferences and domain knowledge," *Automated Softw. Eng.*, vol. 26, no. 1, pp. 87–123, 2019.
[8] J. A. Pereira, "A collaborative-based recommender system for configuration of extended product lines," in *Proc. 39th Int. Conf. Softw. Eng. Companion*, 2017, pp. 445–448.
[9] J. A. Pereira, P. Matuszyk, S. Krieter, M. Spiliopoulou, and G. Saake, "A feature-based personalized recommender system for product-line configuration," in *Proc. ACM SIGPLAN Int. Conf. Generative Program., Concepts Exper.*, 2016, pp. 120–131.
[10] J. A. Pereira, P. Matuszyk, S. Krieter, M. Spiliopoulou, and G. Saake, "Personalized recommender systems for product-line configuration processes," *Comput. Lang., Syst. Struct.*, vol. 54, pp. 451–471, Dec. 2018.
[11] A. B. Sánchez, S. Segura, and A. Ruiz-Cortés, "The Drupal framework: A case study to evaluate variability testing techniques," in *Proc. 8th Int. Workshop Variability Modelling Softw.-Intensive Syst. (VAMOS)*, Nice, France, Jan. 2014, Art. no. 11.
[12] J. A. Galindo, H. Turner, D. Benavides, and J. White, "Testing variability-intensive systems using automated analysis: An application to android," *Softw. Qual. J.*, vol. 24, no. 2, pp. 365–405, 2016.
[13] J. A. Pereira, J. Martinez, H. K. Gurudu, S. Krieter, and G. Saake, "Visual guidance for product line configuration using recommendations and non-functional properties," in *Proc. 33rd Annu. Symp. Appl. Comput. (SAC)*, Pau, France, Apr. 2018, pp. 2058–2065.
[14] J. A. Pereira, S. Schulze, S. Krieter, M. Ribeiro, and G. Saake, "A context-aware recommender system for extended software product line configurations," in *Proc. 12th Int. Workshop Variability Modelling Softw.-Intensive Syst.*, 2018, pp. 97–104.
[15] J. A. Pereira, S. Schulze, E. Figueiredo, and G. Saake, "N-dimensional tensor factorization for self-configuration of software product lines at runtime," in *Proc. 22nd Int. Syst. Softw. Product Line Conf. (SPLC)*, New York, NY, USA, vol. 1, 2018, pp. 87–97. [Online]. Available: http://doi.acm.org/10.1145/3233027.3233039
[16] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns* (The SEI series in software engineering). Boston, MA, USA: Addison-Wesley, 2001.
[17] T. Thum, C. Kastner, S. Erdweg, and N. Siegmund, "Abstract features in feature modeling," in *Proc. 15th Int. Softw. Product Line Conf. (SPLC)*, 2011, pp. 191–200.
[18] D. Batory, "Feature models, grammars, and propositional formulas," in *Proc. Int. Conf. Softw. Product Lines*. Berlin, Germany: Springer, 2005, pp. 7–20.
[19] J. A. Pereira, K. Constantino, and E. Figueiredo, "A systematic literature review of software product line management tools," in *Proc. Int. Conf. Softw. Reuse*. Cham, Switzerland: Springer, 2015, pp. 73–89.
[20] G. K. Narwane, J. A. Galindo, S. N. Krishna, D. Benavides, J.-V. Millo, and S. Ramesh, "Traceability analyses between features and assets in software product lines," *Entropy*, vol. 18, no. 8, p. 269, 2016.
[21] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Commun. ACM*, vol. 35, no. 12, pp. 61–70, 1992.
[22] G. Linden, B. Smith, and J. York, "Amazon. com recommendations: Item-to-item collaborative filtering," *IEEE Internet Comput.*, vol. 7, no. 1, pp. 76–80, Jan. 2003.
[23] A. Tuzhilin, Y. Koren, J. Bennett, C. Elkan, and D. Lemire, "Large-scale recommender systems and the netflix prize competition," in *Proc. KDD*, 2008, pp. 1–34.
[24] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 734–749, Jun. 2005.
[25] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, "GroupLens: Applying collaborative filtering to Usenet news," *Commun. ACM*, vol. 40, no. 3, pp. 77–87, 1997.
[26] J. B. Schafer, J. Konstan, and J. Riedl, "Recommender systems in e-commerce," in *Proc. 1st ACM Conf. Electron. Commerce*, 1999, pp. 158–166.
[27] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. 10th Int. Conf. World Wide Web*, 2001, pp. 285–295.
[28] P. Melville and V. Sindhwani, "Recommender systems," in *Encyclopedia of Machine Learning*. Boston, MA, USA: Springer, 2011, pp. 829–838.
[29] S. K. Lam and J. Riedl, "Shilling recommender systems for fun and profit," in *Proc. 13th Int. Conf. World Wide Web*, 2004, pp. 393–402.
[30] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An open architecture for collaborative filtering of netnews," in *Proc. ACM Conf. Comput. Supported Cooperat. Work*, 1994, pp. 175–186.
[31] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.
[32] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of dimensionality reduction in recommender system—A case study," Dept. Comput. Sci., Univ. Minnesota, Minneapolis, MN, USA, Tech. Rep., 2000.
[33] M. W. Berry, S. T. Dumais, and G. W. O'Brien, "Using linear algebra for intelligent information retrieval," *SIAM Rev.*, vol. 37, no. 4, pp. 573–595, 1995.
[34] M. Pazzani and D. Billsus, "Learning and revising user profiles: The identification of interesting Web sites," *Mach. Learn.*, vol. 27, no. 3, pp. 313–331, 1997.
[35] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *Adapt. Web*. Berlin, Germany: Springer, 2007, pp. 325–341.
[36] S. K. Patel, V. R. Rathod, and S. Parikh, "Joomla, drupal and wordpress—A statistical comparison of open source CMS," in *Proc. 3rd Int. Conf. Trendz Inf. Sci. Comput. (TISC)*, 2011, pp. 182–187.
[37] A. Gunawardana and G. Shani, "Evaluating recommender systems," in *Recommender Systems Handbook*. Boston, MA, USA: Springer, 2015, pp. 265–308.
[38] S. Geisser, *Predictive Inference*. Evanston, IL, USA: Routledge, 2017.
[39] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. Int. Joint Conf. AI*, Aug. 1995, vol. 14. no. 2, pp. 1137–1145.
[40] L. Lü, M. Medo, C. H. Yeung, Y.-C. Zhang, Z.-K. Zhang, and T. Zhou, "Recommender systems," *Phys. Rep.*, vol. 519, no. 1, pp. 1–49, 2012.
[41] J. A. P. Maestre, J. García, A. Ruiz-Cortés, and J. C. Riquelme, "Statservice: Herramienta de análisis estadístico como soporte para la investigación con metaheurísticas," *Actas del VIII Congreso Expañol sobre Metaheurísticas, Algoritmos Evolutivos y Bio-inspirados*, vol. 2012, 2012. [Online]. Available: http://moses.us.es/statservice/
[42] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced non-parametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, 2010.
[43] R. Van Meteren and M. Van Someren, "Using content-based filtering for recommendation," in *Proc. Mach. Learn. New Inf. Age, MLnet/ECML2000 Workshop*, 2000, pp. 47–56.
[44] M. Al-Hajjaji, T. Thüm, J. Meinicke, M. Lochau, and G. Saake, "Similarity-based prioritization in software product-line testing," in *Proc. 18th Int. Softw. Product Line Conf.*, vol. 1, 2014, pp. 197–206.
[45] R. Mazo, C. Dumitrescu, C. Salinesi, and D. Diaz, "Recommendation heuristics for improving product line configuration processes," in *Recommendation Systems in Software Engineering*. Berlin, Germany: Springer, 2014, pp. 511–537.
[46] J. Martinez, G. Rossi, T. Ziadi, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Estimating and predicting average likability on computer-generated artwork variants," in *Proc. Companion Publication Annu. Conf. Genetic Evol. Comput.*, 2015, pp. 1431–1432.
[47] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver, "Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering," in *Proc. 4th ACM Conf. Recommender Syst.*, 2010, pp. 79–86.
[48] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2008, pp. 426–434.
[49] F. Ricci, L. Rokach, and B. Shapira, *Introduction to Recommender Systems Handbook*. Boston, MA, USA: Springer, 2011.

JORGE RODAS-SILVA received the M.Sc. degree in information and communication technology from the University of Milagro, Ecuador, in 2010. He is currently pursuing the Ph.D. degree in computer engineering with the University of Seville, Spain. He has been an Associate Professor with the University of Milagro, since 2012. He has a work trajectory on development of enterprise information systems for decision making. His primary areas of interests are recommender systems, software product lines, business intelligent, and development of technological tools applied to software engineering.

JOSÉ A. GALINDO received the Ph.D. degree (Hons.) from the University of Seville and the University of Rennes 1, in 2015, receiving the award for the Best National Thesis by SISTEDES. He has developed his post-doctoral research activity with INRIA, France. He is currently a Juan de la Cierva Researcher with the University of Seville, where he continues his line of research on configuration, testing, and the evolution of highly configurable systems. He has developed his professional activity in USA, France, and Spain. His research areas are product lines software and the configuration of such products.

JORGE GARCÍA-GUTIÉRREZ received the Ph.D. degree in computer engineering from the University of Seville, Spain, in 2012. He has been with the Department of Computer Science, University of Seville, since 2008, where he is currently a Lecturer Professor. His primary areas of interests are machine learning techniques, big data, remote sensing, data fusion, and evolutionary computation.

DAVID BENAVIDES received the B.S. degree in information systems from the Institute Superieur d'Electronique de Paris, France, in 2000, the M.Sc. degree in computer engineering from the University of Seville, Spain, in 2001, and the Ph.D. degree in software engineering from the University of Seville, in 2007, where he has been an Associate Professor, since 2010. His main research interests include software product line and artificial intelligence applied to engineering education.