Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 4984375, 16 pages
https://doi.org/10.1155/2020/4984375

Research Article

HyOASAM: A Hybrid Open API Selection Approach for Mashup Development

Bo Jiang, Pengxiang Liu, Ye Wang, and Yezhi Chen

School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou 310018, China

Correspondence should be addressed to Ye Wang; yewang@zjgsu.edu.cn

Received 30 December 2019; Revised 20 March 2020; Accepted 1 April 2020; Published 25 April 2020

Guest Editor: Hua Ming

At present, Mashup development has attracted much attention in the field of software engineering. It is the focus of this article to use existing open APIs to meet the needs of Mashup developers. Therefore, how to select the most appropriate open API for a specific user requirement is a crucial problem to be solved. We propose a Hybrid Open API Selection Approach for Mashup development (HyOASAM), which consists of two basic approaches: one is a user-story-driven open API discovery approach, and the other is multidimensional-information-matrix- (MIM-) based open API recommendation approach. The open API discovery approach introduces user stories in agile development to capture Mashup requirements. First, it extracts three components from user stories, and then, it extracts three corresponding properties from open API descriptions. Next, the similarity calculation is performed on two sets of data. The open API recommendation approach first uses MIM to store open APIs, Mashups, and the invoking relationship between them. Second, it enters the matrix obtained in the previous step into a factorization machine model to calculate the association scores between the Mashups and the open APIs, and TOP-N open API lists for creating the Mashup are obtained. Finally, experimental comparison and analysis are carried out on the PWeb dataset. The experimental results show that our approach has improved significantly.

1. Introduction

Unlike object-oriented software engineering [1, 2], service-oriented software engineering (SOSE) is used to design, develop, and maintain software systems [3] that use the principles of service-oriented architecture (SOA) [4]. Open APIs are the basis of SOSE [5]. Mashup development is a novel development practice of SOSE for building multiservice applications by integrating single-function open APIs, which becomes more and more popular. Mashup refers to a temporary combination of web applications that allows users to create entirely new APIs using content retrieved from external data sources [6]. As Mashup application developers face the explosive growth of open APIs on the Internet, they often suffer from the overload of API information.

At present, there are a large number of open APIs over the Internet with similar descriptions but different functionalities and qualities, which undoubtedly affect Mashup application developers' decisions. In addition, unstructured description documents of open APIs increase the difficulty of semantic extraction. All of the above problems make it more and more difficult for developers to select the appropriate high-quality open APIs to build Mashup applications. Therefore, a major challenge to Mashup application development has emerged [7]: how to effectively and efficiently select the most appropriate open APIs from a large number of available resources to match the needs of Mashup developers.

In response to the above challenges, a number of open APIs selection approaches have been proposed, including keyword-based discovery approaches [8, 9], topic model-based discovery approaches [10, 11], content-based recommendation approaches [12, 13], and QoS-based recommendation approaches [14, 15]. Yet, there are some problems with these studies: (1) Most established approaches use nonnatural language (NL) text to describe Mashup requirements [16], such as WSDL, which is not user friendly. The existing techniques do not work well with text modeling. (2) Some users do not even know how to describe the

requirements of Mashup exactly [17], which makes the API search based on keywords difficult. Therefore, a comprehensive open API selection approach should not only include searching or discovering APIs based on keywords but also include actively recommending APIs to developers based on his/her preference. Yet, current approaches separate open API discovery from open API recommendation, leading to inefficient results.

To overcome the above problems, we propose a Hybrid Open API Selection Approach for Mashup development (HyOASAM), which consists of two basic approaches: one is a user-story-driven open API discovery approach, and the other is a multidimensional-information-matrix- (MIM-) based open API recommendation approach. The user-story-driven open API discovery approach is to tackle the first problem. By introducing user stories into Mashup development, the Mashup developers can easily capture the role, aim, and motivation of a Mashup and then describe them with NL-based user stories. In agile development, user stories are used to capture and describe the rapidly changing user requirements. The MIM-based open API recommendation approach is to tackle the second problem. We make use of the historical information of Mashup developers (such as the search history and the access history) to profile each developer, elicit their preferences, and then recommend the most suitable open APIs to them. The open API discovery approach can be divided into three steps: (1) extracting three components from user stories, (2) extracting three corresponding elements from open API descriptions, and (3) calculating the similarity based on two sets of data. The open API recommendation approach can be divided into two steps: (1) extracting open APIs, Mashups, and the invoking relationships between them using MIM and (2) calculating the association scores between the Mashup and the open APIs using a factorization machine (FM) model to recommend TOP-N open APIs. Our approach can perform attribute extraction well for both long and short texts, and deeper associations can be better mined through FM.

In summary, the contributions of this paper are as follows:

(1) We propose a novel hybrid open API selection approach, HyOASAM, enabling developers to quickly and accurately find their wanted open APIs by both discovering APIs and recommending APIs.

(2) We propose an approach that breaks the restrictions of open API documents, described by user stories, which can be used to capture and describe Mashup developers' requirements more accurately and effectively.

(3) We tailor the current MIM matrix [26] and introduce factorization machine (FM) into the open API recommendation approach to calculate the semantic similarity more accurately.

(4) We validate the effectiveness of HyOASAM through various evaluation criteria based on the real data of PWeb. The evaluation results show that HyOASAM has a better improvement over other approaches.

The rest of the paper is organized as follows: Section 2 introduces previously established open API discovery and recommendation approaches; Section 3 introduces our proposed HyOASAM in detail; Section 4 compares the HyOASAM with other approaches; Section 5 draws the conclusions.

2. Related Work

In this work, we treat open APIs and services as synonyms and use the two concepts interchangeably.

2.1. Open APIs Discovery. Open APIs discovery is the efficient and accurate retrieval of a set of open APIs or services that achieve the needs of users from a service database based on the demand statements entered by users [18]. Generally, the discovery approaches can be categorized into the following two main classes.

2.1.1. Syntax-Level Discovery Approaches. Syntax-level open API discovery is the earliest proposed discovery technique. It matches through several keywords and the grammatical features of the service interface, and the matching mechanism is relatively simple. Typical approaches are as follows.

Massimo and Erl [8] proposed a solution based on DAML-S (DARPA Agent Markup Language) to perform semantic matching between user requirements and service description. Mateos et al. [19] use metaprogramming and other related technologies to develop a set of tools for text mining and service processing of WSDL documents. Paliwal et al. [20] proposed clustering services by clustering approaches based on service descriptions and service registration information on UDDI and then used latent semantic indexing (LSI) to achieve matching. Elgazzar et al. [21] improved the accuracy of service discovery by searching key information such as content, service types, messages, and ports in the WSDL document and used Quality Threshold (QT) clustering algorithms to group services based on key information.

In general, the syntax-level open APIs discovery approaches are relatively simple to implement and easy to maintain. However, the deep semantics cannot be understood using such approaches. For example, the polysemy is a common problem, which inevitably leads to a low precision.

2.1.2. Semantic-Level Discovery Approaches. Ontology is used to solve the heterogeneity of grammatical-level service descriptions at the early stage, so that the semantic description of services functions and behaviors is strengthened. The matching algorithms in semantic-level service discovery rely on logical deduction and reasoning. The continuous development of artificial intelligence makes the service discovery algorithms smarter, faster, and accurate.

Ke et al. [22] transformed user requirements and service description documents into ontology trees; calculated the conceptual similarity, attribute similarity, and structural similarity of corresponding nodes in a hierarchical and classified manner; and thereafter effectively avoided complex reasoning. Huang et al. [23] proposed a semantic similarity calculation approach based on ontology distance calculation. The AGNES clustering algorithm clustered semantic service sets to improve the efficiency of service discovery. Klusch [24] used service profiles to select services described by OWL-S on a semantic level. Wei et al. [25] proposed a customizable SAWSDL service matcher that extends XQuery by using various similarity measures to support multiple matching strategies based on different application requests.

Most of the above open API discovery approaches cannot meet the dynamic changing needs of Mashup developers to provide service choices. Moreover, most of the Mashup developers' requirement descriptions are inaccurate and cannot describe their real needs better. This will greatly affect the results of service discovery. In traditional topic-based data extraction models, such as LDA, TF-IDF, HDP, and other topic models, the topic extraction method is the generalization of the user's overall needs, and the extracted data will have a certain deviation from the actual needs of the user; that is, it cannot describe user preferences and needs.

2.2. Open APIs Recommendation. With the rapid increase of open APIs, some APIs that are of interest to Mashup developers are difficult to search because of the small number of visits. On the other hand, developers often lack reasonable and effective requirement description skills. In such cases, open APIs discovery cannot be applied appropriately. Open APIs recommendation maintains the ecology of the service platform to tackle the problem. At present, the existing recommendation research work can be roughly divided into the following three categories.

2.2.1. Recommendation Based on Functional Characteristics. The functional characteristics are mainly extracted from service description documents, and the most similar services are recommended by measuring the similarity between the description documents. For example, Cao et al. [26] used the topic model to calculate the relationship between Mashup, services, and the invoking relationship between them. By integrating the popularity of the service into the model, they predicted the link between the Mashup and the service and then recommended the appropriate service for the Mashup developer. Most service recommendation approaches based on functional characteristics adopt traditional topic models or keywords for similarity calculation. However, traditional topic models need to specify the number of topics in advance while extracting topic vectors, which has a direct impact on the recommendation results.

2.2.2. Recommendation Based on Quality of Service (QoS). QoS refers to the nonfunctional features of the service, such as the user's history of invoking services or the quality of services. Zheng et al. [27] used collaborative filtering to calculate the quality of services through user historical behaviors. Huang et al. [28] proposed a Mashup component recommendation approach to establish a relationship between Mashup components through a generic layer model and guide users to select components from a large-scale Mashup component library. Xu et al. [29] proposed a socially perceived approach, in which the coupling matrix model was used to store the multidimensional social network between potential users, topics, Mashups, and services, and the relationships were predicted by existing relational networks. However, those approaches have the problem of matrix sparsity, which affects the recommendation accuracy.

2.2.3. Recommendation Based on Hybrid Characteristics. Such approaches take into consideration not only the functional characteristics of the service but also the QoS. By combining the two characteristics, the accuracy of service recommendation is improved. For example, Gao et al. [30] proposed a manifold ordering framework. Based on the similarity between Mashups and the heterogeneous relationship between Mashups and services, a manifold ranking algorithm is applied to recommendation services. The similarity between Mashups and the heterogeneous relationship between Mashups and services are calculated by a manifold sorting algorithm. Li et al. [31] proposed an approach for integrating multidimensional information, using HDP to extract service, and the subject vector of Mashups was used to calculate the similarity between Mashups, the similarity between services, the fluency of services, and the co-occurrence of services. Then they used the FM model to score and recommend the highest rated N services to Mashup developers. Xia et al. [32] proposed a new class-aware service clustering and distributed approach. First, the services were clustered by extending the K-means clustering algorithm, and then, the service ordering was predicted through a distributed machine learning framework. At present, the service recommendation approaches based on hybrid characteristics are among research hotspots, due to its high precision. HyOASAM has taken the advantage of such approaches.

From Table 1, we can see that the established open APIs discovery approaches have the problem of the randomness of user demand descriptions and open APIs description texts, which leads to unsatisfactory results, whereas open APIs recommendation approaches have the problem of inability to fully mine hidden information. Besides, it is hard to meet the real needs of developers to take either the discovery approaches or the recommendation approaches. A hybrid manner is a more accurate and comprehensive manner.

3. HyOASAM Approach

If a Mashup developer enters a user story of a requirement "as a user, I want to upload and edit photos online so that I can process photos on the server," then user-story-driven open APIs discovery approach is applied to calculate the similarity

Table 1: The comparison of related work.

Approach	Category	Pros	Cons
Service discovery based on DAML-S [8]	Open API discovery	The earliest service described by DARPA agent markup language	The randomness of user demand descriptions and service description texts leads to unsatisfactory results
Service discovery based on text mining [19]		It combines text mining and metaprogramming techniques	The approach is unable to mine deep relationships
Web service discovery based on an ontology [20, 22, 24]		They address the issue of nonexplicit service description semantics that match a specific service request	The semantic extension is not enough
Web service discovery based on WSDL documents clustering [21]		Narrowing the search space and improving results	Each feature is not assigned its own weight
Web service discovery based on hierarchical clustering [23]		The vector space model improves the accuracy and efficiency	It does not take the semantics into consideration
Web service discovery based on SAWSDL-iMatcher [25]		Multiple matching strategy extensions via XQuery can effectively aggregate similar values	The approach is only useful in one specific domain, not effective in other domains
Open API recommendation based on topic model [26, 31]	Open API recommendation	The document probability distribution is obtained, and the distance is used to calculate the semantic distance	The topic model is not well trained
Web service recommendation based on collaborative filtering [27]		Collaborative filtering does not require specialized domain knowledge and can be easily modeled	Collaborative filtering cannot mine hidden information
Model-based recommendation [28]		The use of a generic hierarchical graph model can improve efficiency and effectiveness	This approach cannot get synthesis of multiple constraints for more personalized recommendation
Social-aware recommendation [29]		It can predict unobserved relationships	The matrix sparsity affects accuracy
Manifold-learning-based recommendation [30]		Mashup can use manifold sorting algorithm for better clustering	The approach cannot handle dynamically added services
Combining machine learning and distributed recommendation [32]		More accurate prediction	The approach ignores QoS
HyOASAM	Open API discovery and recommendation	HyOASAM can handle random description text and make accurate recommendations for unclear user needs description	The modeling process is a little more complicated than other approaches

between the requirement and the open API description text and returns a list of open APIs for developers according to the level of similarities. If no requirements are entered, the MIM-based open API recommendation approach is applied to calculate the score between the developer's profiles including the Mashup usage and the information of the open APIs and returns a list of open APIs for developers to choose from based on the score. Below we will use the example to illustrate the whole process of HyOASAM.

The framework of the HyOASAM approach is shown in Figure 1. HyOASAM takes two scenarios into account: (1) When the Mashup developer can describe his/her requirements with user stories, the user-story-driven open API discovery approach is applied to return a list of open APIs to the developer. (2) When the Mashup developer does not input any requirement, the MIM-based open API recommendation is applied to return a list of open APIs to the developer.

3.1. User-Story-Driven Open APIs Discovery Approach. The user-story-driven open APIs discovery approach analyzes syntactic dependencies [33] of user stories by the Mashup developer's requirements and then uses NLP technology to extract the requirements components (Step 1). At the same time, open API properties are extracted in the open API description (Step 2). Then, it calculates the similarity between the requirement components and open API properties. Finally, it sorts the open APIs by the similarity to get the open API list with the most similar N for developers to choose from (Step 3). Figure 2 shows the overall framework of the open API discovery approach.

3.1.1. Requirements Components Extraction (Step 1)

Definitions. Developer requirement descriptions are often too casual, so we use user story to describe open API requirement components [34]. For example, "as a user, I want to upload and edit photos online so that I can process photos on the server". Requirement components are the key information of open API requirements, and the detailed description is shown as follows:

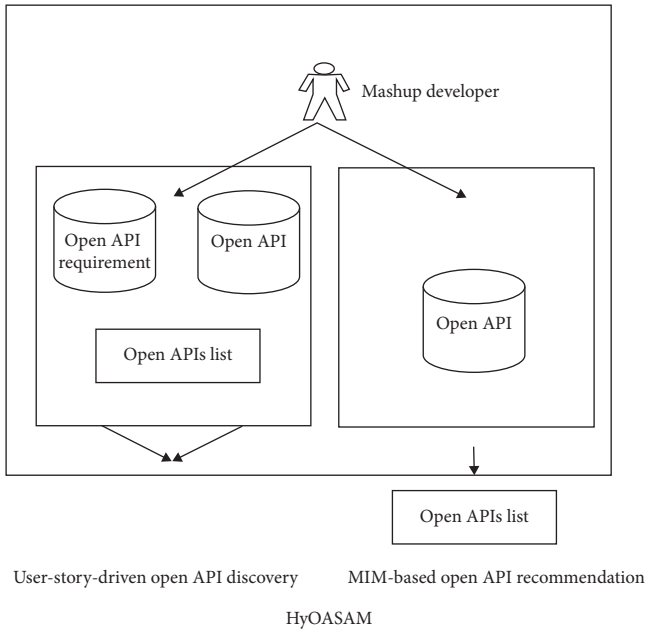

Figure 1: Framework of HyOASAM.

(1) Role (ro) = <noun, adj>: the role to carry out the functionality. The noun is the role, and the adj is a modification of the noun. In the above example, the role is <user, null>.

(2) Aim (ai) = <verb, do, io, adj>: the aim that developers want to achieve. Verb is act of a role, do is the direct object, io is the indirect object, and adj is the extension of the corresponding noun. In the above example, the aim is <{upload, edit}, photo, null, online>.

(3) Motivation (mo) = <verb, do, io, adj>: the developers' purpose. Components in mo and go are the same. In the above example, the motivation is <process, photo, server, null>.

Definition 1 (requirements components). rc = <ro, ai, mo>. Requirements component represents developers' actual needs and is composed of role, aim, and motivation.

In the process of requirements components extraction, we tag each word in the user story, because polysemy will affect the final result. Then, we extract requirement components based on grammatical dependencies. We use Stanford Parser [35] to parse text and extract Stanford Dependency (SD) set. Proceed as follows:

(i) *Role extraction*. We have found through several experiments that the following three SDs can completely extract the components of role, including pobj(As, dep), nn(gov, dep), and amod(gov, dep). Readers can refer to the white paper of Stanford Parser [35] for the detailed explanation of each SD.

(ii) *Aim extraction*. The following eight SDs and their combinations can extract all aim components, including xcomp(want, dep), dobj(gov, dep), iobj(gov, dep), conj(gov, dep), pobj(gov, dep), nn(gov, dep), and amod(gov, dep).

(iii) *Motivation extraction*. As aim and motivation have similar structures, their components extraction processes are similar. We only need to change the first SD here into aux(dep, can).

For example, when the Mashup developer enters the requirement "as a user, I want to upload and edit photos online so that I can process photos on the server," through Step 1, the final extraction result of the requirement component is like this: <user, null>, <{upload, edit}, photo, null, online>, <process, photo, server, null>.

3.1.2. Open API Properties Extraction (Step 2). Next, we extract open API properties from the open API description text. The open API description text is generally a text describing the API function written by the API developer. It is mainly a text that helps the developer understand the API and how to use it. Currently, the open API description text is written in NL, for example, "customers can use the service to edit photos and video over the Internet." Open API properties contain the following properties:

(1) Agent (ag) = <noun, adj>. The subject that the open API is served to. In the above example, the agent is <customer, null>.

(2) Activity (ac) = <verb, do>. The activity provided by the open API. In the above example, the activity is <edit, {photo, video}>.

(3) Scenario (sc) = <io, adj>. The scenario of the open API. In the above example, the scenario is <Internet, over>.

Definition 2 (open APIs properties). oap = <ag,ac,sc> represents the properties of the entire open APIs.

As the style of the open API description text is not limited, we extract the following 14 SDs to comprise open API properties: nsubj, nsubjpass, xsubj, agent, csubj, csubjpass, cop, nn, dobj, iobj, prep&pobj, pobj, amod, and conj.

For example, the open API description text is "customers can use the service to edit video and photos over the Internet." Through Step 2, the final result of open API properties is <customer, null>, <edit, {photo, video}>, <Internet, over>.

3.1.3. Similarity Calculation (Step 3). This section presents the similarity formula between the user story q and open API s through requirements components and open APIs properties.

The overall formula is as follows:

$$\begin{aligned} \mathrm{sim}(q,s) = a \times \mathrm{usim}\left(u_q, u_s\right) + b \times \mathrm{asim}\left(a_q, a_s\right) \\ + c \times \mathrm{gsim}\left(g_q, g_s\right), \end{aligned} \tag{1}$$

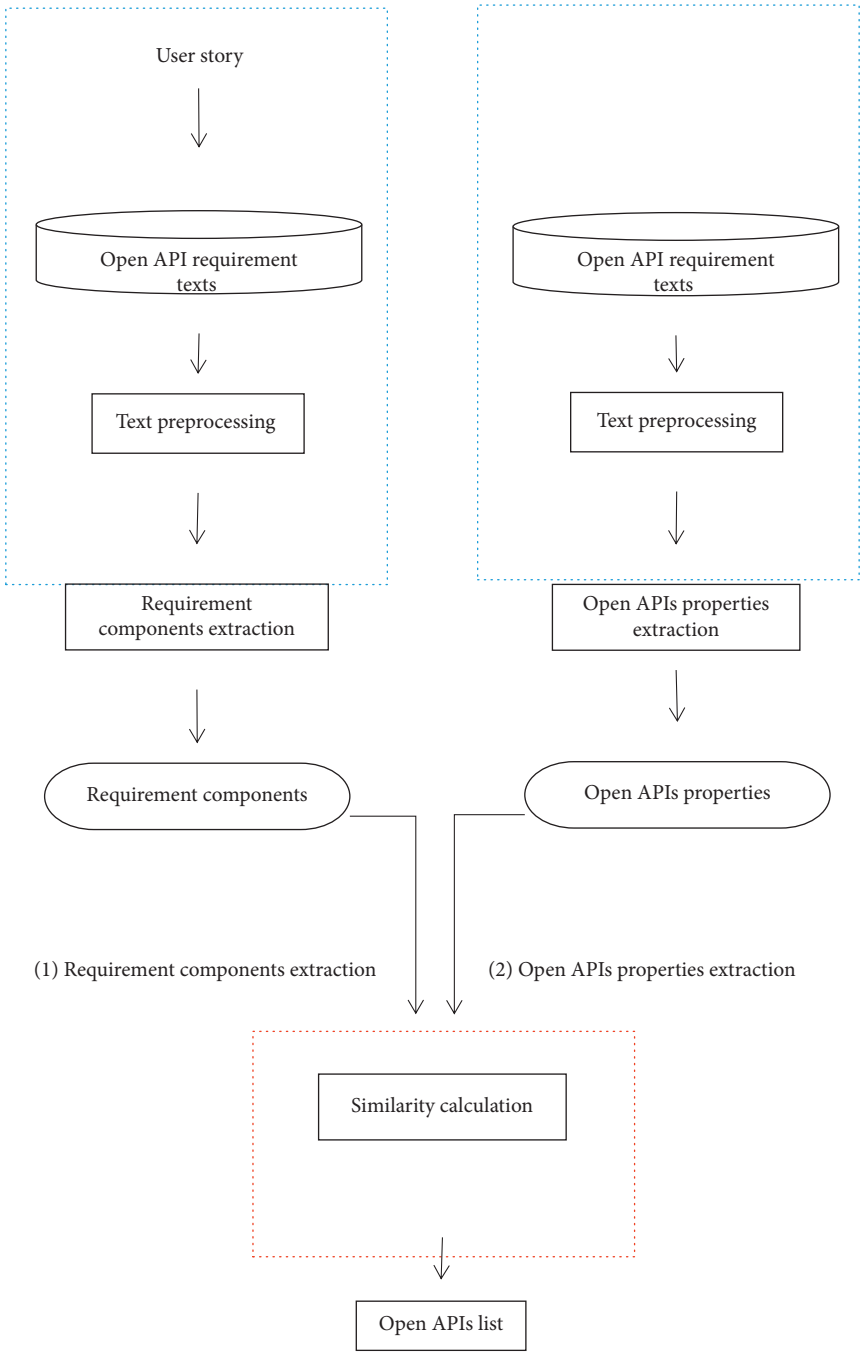

Figure 2: Framework of the user-story-driven open API discovery approach.

where $\mathrm{usim}(u_q, u_s)$ represents the similarity between the role components u_q in user story q (e.g., <user, null>) and the agent properties u_s in open API description text s (e.g., <customer, null>); $\mathrm{asim}(a_q, a_s)$ represents the similarity between part of aim and motivation components a_q (verb and do) in user story q (e.g., <{upload, edit}, photo>,<process, photo>) and the activity properties a_s (verb and do) in open API description text s (e.g., <edit, {photo, video}>); $\mathrm{gsim}(g_q, g_s)$ represents the similarity between part of aim components g_q (io and adj) in user story u (e.g., <null, online><server, null>) and scenario properties g_s (io and adj) in open API description text s (e.g., <Internet, over>). The parameters a, b, and c represent the weight of the three variables in

(1), and $a+b+c=1$. The specific formula is as follows:

$$\mathrm{usim}\left(u_q, u_s\right) = \begin{bmatrix} \mathrm{sim}\left(w_{q_1} w_{s_1}\right) & \cdots & \mathrm{sim}\left(w_{q_1} w_{s_j}\right) \\ \vdots & \ddots & \vdots \\ \mathrm{sim}\left(w_{q_k} w_{s_1}\right) & \cdots & \mathrm{sim}\left(w_{q_k} w_{s_j}\right) \end{bmatrix}, \quad (2)$$

$$\mathrm{gsim}\left(g_q, g_s\right) = \begin{bmatrix} \mathrm{sim}\left(w_{q_1} w_{s_1}\right) & \cdots & \mathrm{sim}\left(w_{q_1} w_{s_j}\right) \\ \vdots & \ddots & \vdots \\ sim\left(w_{q_k} w_{s_1}\right) & \cdots & sim\left(w_{q_k} w_{s_j}\right) \end{bmatrix}, \quad (3)$$

$$\mathrm{sim}\left(w_{q_k}, w_{s_j}\right) = \frac{\vec{w}_{q_k} \vec{w}_{s_j}}{\| w_{q_k} \| \| w_{s_j} \|}. \quad (4)$$

In (2), K is the amount of words in u_q and J is the amount of words in u_s. Equation (3) is the same as (2). In (4), first, we vectorize the word with Word2Vec [36] and then calculate the similarity. w_{q_k} represents the word vector corresponding to Word2Vec:

$$\alpha_{k,j} = \frac{\exp\left(\mathrm{sim}\left(w_{q_k}, w_{s_j}\right)\right)}{\sum_{i=1}^{j} \exp\left(\mathrm{sim}\left(w_{q_k}, w_{s_j}\right)\right)}. \quad (5)$$

We normalize the similarity of each row to calculate the weight $\alpha_{k.j}$:

$$u_k = \sum_{j=1}^{J} \alpha_{k,j} \mathrm{sim}\left(w_{q_k}, w_{s_j}\right). \quad (6)$$

We use the accumulation of the multiplication of weights and similarities as the similarity of each row in u:

$$\mathrm{usim}\left(u_q, u_s\right) = \frac{1}{k} \sum_{i=1}^{k} u_i,$$

$$\mathrm{gsim}\left(g_q, g_s\right) = \frac{1}{k} \sum_{i=1}^{k} u_i. \quad (7)$$

The formula for $\mathrm{asim}\left(a_q, a_s\right)$ is as follows:

$$\mathrm{asim}\left(a_q, a_s\right) = \frac{\sum_{i=1}^{n} \max\left(\mathrm{masim}\left(a_{q_i}, a_{s_{1\sim m}}\right)\right)}{n}. \quad (8)$$

Here n is the amount of verbs in the aim and motivation of u and m is the amount of verbs in the activities of s. $\mathrm{masim}\left(a_{q_i}, a_{s_i}\right)$ represents the similarity between an element in a_q and an element in a_s:

$$\mathrm{masim}\left(a_q, a_s\right) = w_1 \times \mathrm{sim}\left(V_q, V_s\right) + w_2 \times \frac{\sum_{i=1}^{I} \max\left(\sum_{j=1}^{J} \mathrm{sim}\left(N_{q_i}, N_{s_j}\right)\right)}{I}. \quad (9)$$

In (9), V_q is a verb in aim or motivation, V_s is another verb in activity, I and J represent the number of nouns in a_q and a_s, respectively, N_{q_i} is the i-th noun in a_q, N_{s_j} is the j-th noun in a_s, and w_1 and w_2 are the weights of verbs and nouns, respectively. Using (4), $\mathrm{sim}\left(V_q, V_s\right)$ and $\mathrm{sim}\left(N_{q_i}, N_{s_j}\right)$ are calculated as similarities between words.

For example, when calculating the similarity between requirements and the open API description text in step 3, $\mathrm{usim}\left(u_q, u_s\right)$ and $\mathrm{gsim}\left(g_q, g_s\right)$ are calculated in the same way. Here, we take $\mathrm{gsim}\left(g_q, g_s\right)$ as an example. At this time, g_q is <null, online><server, null>, and g_s is <Internet, over>. First establish the similarity matrix $\begin{bmatrix} \mathrm{sim}(\mathrm{null}, \mathrm{Internet}) & \mathrm{sim}(\mathrm{null}, \mathrm{over}) \\ \mathrm{sim}(\mathrm{server}, \mathrm{Internet}) & \mathrm{sim}(\mathrm{server}, \mathrm{over}) \\ \mathrm{sim}(\mathrm{online}, \mathrm{Internet}) & \mathrm{sim}(\mathrm{online}, \mathrm{over}) \\ \mathrm{sim}(\mathrm{null}, \mathrm{Internet}) & \mathrm{sim}(\mathrm{null}, \mathrm{over}) \end{bmatrix}$ of gsim, where sim(online, Internet) is calculated using Word2Vec similarity. After obtaining the similarity matrix, calculate the $a_{k,j}$ weight for every similarity of each row; the weight indicates the importance of each element in each row in the entire row; then, linearly combine the weight with the corresponding elements, and finally take the similarity of all rows; the average of the values is taken as the final similarity of gsim.

Compared with the calculation of gsim, asim needs to calculate the similarity between verbs and verbs, nouns and nouns, and then perform linear combination of weights. Specifically, first calculate the similarity masim (a_q, a_s) between an element in a_q and an element in a_s; a_q is composed of aim and motivation (io and adj), that is, <{upload, edit}, photo> and <process, photo>; a_s is composed of scenario (io and adj), that is, <edit, {photo, video}>. Because a_q has 3 verbs and a_s has only one, $n=3$ and $m=1$. First calculate the similarity between < upload, photo> and <edit, {photo, video}>; the verb upload and verb edit are calculated with Word2Vec, for the similarity of the nouns: photo and photo, video; because < upload, photo > has only one noun in a_q, <edit, {photo, video}> in a_s has two nouns, $I=1$, $J=2$; calculate sim(photo, photo) and sim(photo, video) respectively; take the maximum value of the similarity of a noun k in a_q to all nouns in a_s as the similarity of k to a_s nouns, and then take the average value of the similarity of all nouns in a_q to the a_s noun as the noun similarity between a_q and a_s. Finally, the weight of the verb and the similarity of the verb, and the weight of the noun and the similarity of the noun are linearly combined to obtain the final similarity, that is, the final masim(a_q, a_s). Similarly, calculate the similarity of <edit, photo> and <edit, {photo, video}>, <process, photo> and <edit, {photo, video}>. Take the maximum value of the similarity of an element in a_q to each element in a_s as the similarity of the a_q element to a_s, and at this time $m=1$, so the similarity of each element in a_q to a_s is the maximum. $n=3$. Finally, calculate the average similarity of all elements in a_q to a_s as the final gsim.

Linearly combine usim, asim, and gsim to get the similarity between the final user story and the open API description text.

3.2. MIM-Based Open API Recommendation Approach. Usually, there are a lot of existing open APIs and Mashups in the API registration platforms. Inspired by Li et al. [31], we make use of these existing open APIs and Mashups as well as

their properties as multiple-dimensional information to recommend open APIs to Mashup developers when these developers cannot describe their requests clearly. In Figure 3, the MIM-based open API recommendation approach is generally divided into two steps:

Step 1. MIM construction. The MIM contains an active open API matrix, a target Mashup matrix, a similar open API matrix, a similar Mashup matrix, a co-occurrence matrix, and a popularity matrix. The active open APIs matrix is the open API currently used for scoring prediction, and the target Mashup matrix is the Mashup served by the recommended method. First, to build the similar open API and similar Mashup matrix, we extract three elements (i.e., agent, activity, scenario) of open APIs properties and two elements (i.e., Mashup activity, Mashup scenario) of Mashup properties separately by the dependency syntax relationship, so as to combine these elements, respectively, into a connected network to characterize the relationships. Then, the co-occurrence of open APIs is the external manifestation of open API composition relationships. Finally, the QoS property, i.e., the open API popularity, is measured by the open API category and the historical invoking times.

Step 2. FM model score prediction. Different types of information such as the open API popularity, the connection model, and the co-occurrence are trained by the FM to calculate the interaction between Mashups and open APIs, obtain corresponding prediction scores, and then return the list of TOP-N open APIs with the highest score to Mashup developer.

3.2.1. MIM Matrix Construction (Step 1). As mentioned before, MIM is an integrated matrix including multiple-dimensional information. In MIM, the active open APIs matrix represents the open API currently used for the score prediction. The target Mashup matrix represents the Mashup to be developed. The similar open API matrix presents the similarities between open APIs. The similar Mashup matrix represents the similarity between the Mashup in the Mashups matrix and their similar Mashups. The co-occurrence matrix represents the co-occurrence relationships between the active open APIs and other open APIs. The popularity matrix represents the total frequency of the invoking history of the active open APIs.

(1) The Similar Open APIs Matrix Construction. Open API properties extraction. Open APIs properties are selected as the functional characteristics for open API recommendation, i.e., the agent, activity, and scenario. The extraction process is the same as what we have shown in Section 3.1 and will not be described here.

Similarity calculation: the calculation of similarity is the same as what we have shown in Section 3.1.3 and will not be described here. A similar open APIs matrix can be obtained by multiple calculations of the above similarity approach.

(2) The Similar Mashup Matrix Construction. Mashup properties extraction. Two Mashup properties are extracted from the Mashup description text and the number of identical open APIs invoked between Mashups.

(1) Mashup description text extraction:

The Mashup description text recorded the detailed information of the Mashup and a storage carrier, which is written in NL and in any format, for example, "customers can use the service to upload and edit photos over the Internet." We randomly extract 10,000 Mashup application descriptions from the application library and extract SDs on the application descriptions. Compared with open API, Mashup lacks user description information. Therefore, we only extract the activity (named as Mashup activity) and scenario from the Mashup description text:

(1) Mashup activity (ma): the activity provided by the Mashup, which in the example refers to "upload and edit photos."
(2) Mashup scenario (ms): the scenario of the Mashup, which in the example refers to "over the Internet."

(2) The number of identical open APIs invoked between Mashups

Each Mashup to be developed is composed of two or more open APIs, and different Mashups may invoke the same open API. Therefore, the invoked open APIs can reflect the similarity between two Mashups.

Definition 3 (Mashup properties). mp = <ma, ms>, where ma represents open API key activity information consisting of verbs and nouns. ms represents open API key scenario information consisting of verbs and nouns

Definition 4 (Mashup activity). ma = <verb, object>; the verb is a user-initiated operation. Object exists in the form of a noun, a noun phrase, or a noun phrase in a binary group, such as the open API key activity <{upload, edit}, photo>.

Definition 5 (Mashup scenario). ms = <object>; the verb exists in the form of a verb in a binary group and is a user-initiated operation, <Internet>.

We have identified 10 SDs and classified the SDs into the following six scenarios to extract the Mashup properties, as shown in Table 2.

Similarity calculation: suppose there are two Mashup descriptions "customers can use the service to upload and edit photos over the Internet" and "this Mashup allows users to watch pictures on the server or Internet." We calculate the similarity between Mashups through Mashup description text and the number of identical open APIs invoked between Mashups.

The specific formula for calculating the similarity of two Mashups is shown in

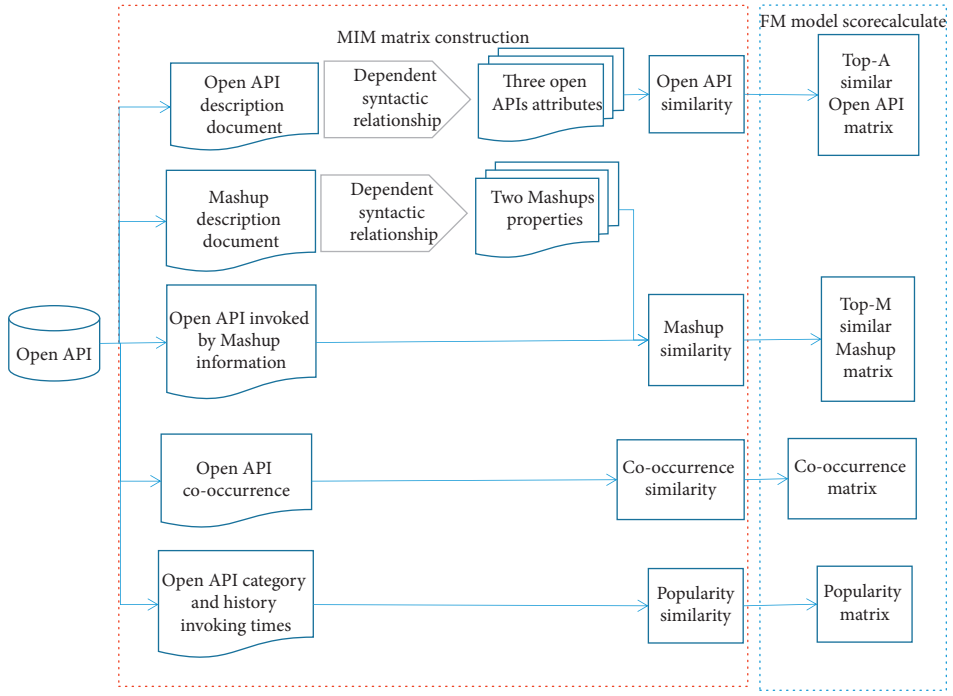

Figure 3: MIM-based open API recommendation approach.

Table 2: Related SDs when extracting Mashup properties.

Scenarios	Related SDs
x, y are placed in the ma verb list	conj(x, y), csubj(x, y), csubjpass(x, y)
x is placed in the ma verb list y is placed in the ma verb list	dobj(x, y), iobj(x, y), prep(x, y) & pobj(y, z) (note: x is a verb)
x is placed in the ma noun list z is placed in the mp noun list	prep(x, y) & pobj(y, z) (note: x is a noun)
y is placed in the ma noun list	nn(x, y) (when x is already in the noun list of ma), cop(x, y)
y is placed in the mp noun list	nn(x, y) (when x is already in the noun list of ms), pobj (x, y)
y is placed in the mp adjective list	amod(x, y)

$$\mathrm{sim}(M_1, M_2) = u \times \mathrm{sim}_{\mathrm{ma}}(M_1, M_2) + v \times \mathrm{sim}_{\mathrm{ms}}(M_1, M_2) + w \times \mathrm{sim}_{\mathrm{se}}(M_1, M_2), \tag{10}$$

where the weight parameter $u = 0.2$, $v = 0.6$, $w = 0.2$ ($u + v + w = 1$). M_1, M_2 represent two different Mashups. $\mathrm{sim}_{\mathrm{ma}}(M_1, M_2)$ represents the similarity of activities between two Mashups (e.g., <{upload, edit}, photo> and <watch, picture>). $\mathrm{sim}_{\mathrm{ms}}(M_1, M_2)$ represents the similarity of the scenarios between two Mashups (e.g., <Internet> and <{server, Internet}>). $\mathrm{sim}_{\mathrm{se}}(M_1, M_2)$ represents the similarity of open APIs invoked between Mashups. The formula of $\mathrm{sim}_{\mathrm{ma}}$ (M_1, M_2) is as follows:

$$\mathrm{sim}_{\mathrm{ma}}(M_1, M_2) = \frac{\sum_{i=1}^{n} \max\left(\mathrm{msim}_{\mathrm{ma}}\left(M_{1_i}, M_{2_{1\sim m}}\right)\right)}{n}, \tag{11}$$

where n is the amount of verbs in the ma of M_1, and m is the amount of verbs in the ma of M_2. masim(a_{q_i}, a_{s_i}) represents the similarity between an element in ma of M_1 and an element in ma of M_2:

$$\mathrm{msim}_{\mathrm{ma}}(M_1, M_2) = w_1 \times \mathrm{sim}(V_1, V_2) + w_2 \times \frac{\sum_{i=1}^{I} \max\left(\sum_{j=1}^{J} sim\left(N_{1_i}, N_{2_j}\right)\right)}{I}. \tag{12}$$

In (11), V_1 is a verb in the ma of M_1, V_2 is another verb in the ma of M_2, I and J represent the number of nouns in the ma of M_1 and ma of M_2, respectively, N_{1_i} is the i-th noun in the ma of M_1, N_{2_j} is the j-th noun in the ma of M_2, and w_1 and w_2 are the weights of verbs and nouns, respectively. Using (4), $\mathrm{sim}(V_1, V_2)$ and $\mathrm{sim}(N_{1_i}, N_{2_j})$ are calculated as similarities between words.

The formula of $\mathrm{Sim}_{\mathrm{ms}}(M_1, M_2)$ is as follows:

$$\mathrm{sim}_{\mathrm{ms}}(M_1, M_2) = \frac{1}{k}\sum_{i=1}^{k} u_i,$$

$$u_i = \begin{bmatrix} \max[\mathrm{sim}(M_{11}, M_{21})] & \cdots & \max[\mathrm{sim}(M_{11}, M_{2j})] \\ \vdots & \ddots & \vdots \\ \max[\mathrm{sim}(M_{1k}, M_{21})] & \cdots & \max[\mathrm{sim}(M_{1k}, M_{2j})] \end{bmatrix}, \tag{13}$$

where u_i represents a set of word combination's similarities. k, j respectively represent the amount of words contained in the ma or ms of the two Mashups M_1 and M_2. $M_{11}, \ldots, M_{1k}$ represent each word corresponding to ma or ms of M_1. $M_{21}, \ldots, M_{2j}$ represent each word corresponding to ma or ms of M_2. The two groups of words are compared one-to-one to calculate the similarity, and a similarity matrix of $k \times j$ is formed. Take the maximum value of the similarity result of each row to represent the similarity of the corresponding words, and finally a matrix of $1 \times k$ is obtained.

We adopted the Jaccard similarity calculation idea to calculate the similarity of open APIs invoked between Mashups:

$$\mathrm{sim}_{\mathrm{se}}(M_{1i}, M_{2j}) = \frac{|A_i \cap A_j|}{|A_i \cup A_j|}. \tag{14}$$

Here, $|A_i \cap A_j|$ represents the amount (the intersection) of the same open API in the corresponding open API composition of the two Mashups, and $|A_i \cup A_j|$ represents the total amount of open APIs (the union) in the open API composition corresponding to the two Mashups.

For example, $\mathrm{sim}(M_1, M_2)$ is composed of three parts: $\mathrm{sim}_{\mathrm{ma}}(M_1, M_2)$, $\mathrm{sim}_{\mathrm{ms}}(M_1, M_2)$, and $\mathrm{sim}_{\mathrm{se}}(M_1, M_2)$. The calculation formula of $\mathrm{sim}_{\mathrm{ma}}(M_1, M_2)$ and $\mathrm{sim}_{\mathrm{ms}}(M_1, M_2)$ is similar to that in Section 3.1.3, which is described in detail here. $\mathrm{sim}_{\mathrm{ma}}(M_1, M_2)$ is the similarity between <{upload, edit}, photo> and <watch, picture>, and $\mathrm{sim}_{\mathrm{ms}}(M_1, M_2)$ is the similarity between <Internet> and <{server, Internet}>. In $\mathrm{sim}_{\mathrm{se}}(M_1, M_2)$, suppose M_1 has invoked a total of 10 open APIs, M_2 has invoked a total of 15 open APIs; both M_1 and M_2 invoke the same 5 APIs, and the final result of $\mathrm{sim}_{\mathrm{se}}(M_1, M_2)$ is $5/(10 + 15) = 0.2$.

Next we linearly combine $\mathrm{sim}_{\mathrm{ma}}(M_1, M_2)$, $\mathrm{sim}_{\mathrm{ms}}(M_1, M_2)$, and $\mathrm{sim}_{\mathrm{se}}(M_1, M_2)$ to get the similarity between M_1 and M_2.

(3) The Co-Occurrence Matrix Construction. Co-occurrence refers to the external connection between open APIs formed in Mashup development. If two different open APIs S_1 and S_2 are directly invoked by the same Mashup, there is co-occurrence between S_1 and S_2. The following formula calculates the co-occurrence between two different open APIs:

$$co(a_i, a_j) = \frac{|a_i \cap a_j|}{|a_i \cup a_j|}. \tag{15}$$

Here, ai and aj represent different open APIs, respectively. $|a_i \cap a_j|$ represents the total times ai and aj were invoked by the same Mashup. $|a_i \cup a_j|$ represents the sum of the times the open API ai and aj were invoked in the past.

(4) The Popularity Matrix Construction. Quality of open API (QoS) is the basis for ensuring the performance of the open API. However, the QoS information is usually vague and dynamic. Therefore, in this work, we have integrated the open API popularity with QoS information to improve the recommendation effect. The open API popularity is calculated by the amount of historical invokes. Therefore, the more popular the domain is, the more invoked the open API in this domain will be. We use (16) to calculate the open API popularity:

$$\mathrm{pop}(ai) = \frac{\mathrm{Fre}(ai) - \min(\mathrm{Fre}(\mathrm{Category}(ai))}{\max(\mathrm{Fre}(\mathrm{Category}(ai)) - \min(\mathrm{Fre}(\mathrm{Category}(ai))}, \tag{16}$$

where Fre(.) calculates the amount of times the corresponding open API has been invoked by the Mashup, Category(.) represents all open APIs that exist in the same domain, max(.) calculates the maximum amount of times the open API has been invoked in history, and min(.) calculate the minimum amount of times the open API has been invoked in history.

3.2.2. FM Model Score Prediction (Step 2). FM can learn the characteristic interaction between Mashup and open API, so as to calculate the correlation information between them. The specific formula of FM is as follows:

$$Y(X) := w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f} x_i\right)^2 - \sum_{j=i+1}^{n} v_{j,f}^2 x_j^2\right), \tag{17}$$

where n is the amount of the features, w_0 is the initial bias term, w_i is the weight of the i-th feature, x_i, x_j represent the interaction between the paired feature variables, $v_{i,f}$, $v_{j,f}$ represent a hidden factor between Mashup x_i and open API x_j in the factorization model, and k is the factorization matrix dimension.

Figure 4 shows an example of an FM model's input and output. The training data consists of two parts. In this work, we use the MIM matrix as input data and a score value Y as output data. Finally, FM can calculate the target value Y between Mashup and the open API and offer recommendations for the Mashup developers by sorting Y.

In the training set, if the active open API is historically invoked by the Mashup, the value in the vector Y is 1; otherwise it is 0. In the test set, the value in vector Y represents the calculated score of the active open API relative to the Mashup. Finally, the final set of recommended open APIs is obtained by ranking the predicted scores.

In the FM model, the model parameters w_0, w, and v are obtained from the training examples. In order to get the optimal parameters, a loss function needs to be defined to obtain the optimal parameter model. We define the loss function as

$$l(\mathrm{Y}(x_i), Y') = \log\left(1 + \exp\left(-\mathrm{Y}(x_i)Y'\right)\right). \tag{18}$$

For example, suppose that there are a total of 3 open APIs and 2 Mashups in the entire dataset. The score between the second open API and the first Mashup is calculated, so the active open API is a vector [0, 1, 0] and the target Mashup is a vector [0, 1], the similar open APIs matrix calculated by the "Similar Open APIs Matrix Construction" section is a vector [0.4, 0, 0.7], the similar Mashup matrix calculated by the "Similar Mashup Matrix Construction" section is a vector [0, 0.3], the co-occurrence matrix calculated by (15) is a vector [0.5, 0], and the popularity matrix calculated by formula (16) is a vector [3]. The final MIM is [0, 1, 0, 1, 0, 0.4, 0, 0.7, 0, 0.3, 0.5, 0, 3] and will be entered into the trained FM to get the final score.

4. Experimental Results and Analysis

We conducted a series of experiments on user-story-driven open API discovery and MIM-based open API recommendation approach to evaluate the effectiveness of the HyOASAM approach [37].

4.1. Experimental Data Collection and Analysis. We crawled all open APIs, Mashups, and information about the relationships between open APIs and Mashups on PWeb. We have collected 15,928 open API items with 398 categories, 6,973 Mashups, and 13,613 links between open APIs and Mashups.

After crawling the data, we performed a series of data processing operations on the data, including filtering special characters and removing open API description texts that are not related to open APIs. After text preprocessing, we selected 1,438 open APIs.

In order to evaluate the effectiveness of HyOASAM, we have established a standard set. We appointed four graduate students with extensive experience in Mashup development to build the establishment of the standard set. The four students built four different sets of open API list standards based on the practical experience they developed. In the end, we used precision as criteria. Due to different standard sets, the final results are also different, and we take the average of the four results as the final standard set of the experiment.

4.2. The Experimental Analysis for Open API Discovery. Three requirements components are extracted from each user story. Table 3 shows the result; we select the five different user stories domains in the table as the experimental open API requirement texts.

4.2.1. Metric Selection. We use precision to evaluate the effectiveness of user-story-driven open API discovery. The precision formula is as follows:

$$\text{precision} = \frac{|S_A \cap S_M|}{|S_A|}. \tag{19}$$

Here, S_A represents the requirements components extracted by HyOASAM and S_M represents the manually extracted open API properties.

4.2.2. Parameter Selection. In (1), a represents the weight of users, b represents the weight of functions, and c represents the weight of motivation. We compared the open API sets from three different domains user stories as input to the three standard open API sets. In Figure 5, it can be seen that the parameters of $a = 0.2$, $b = 0.6$, and $c = 0.2$ are in most cases better than the precision of other parameters. This shows that function is the main factor of the overall similarity.

4.2.3. Comparative Experiment. We compared the user-story-driven open API discovery approach (USDOAD) with other established open API discovery approaches [38, 39]. The two established approaches are as below:

(1) Open API discovery approach based on vector space model (VSMOAD): we used VSM to vectorize the data processed user story $u = \{u_1, u_2, u_3, u_4, \ldots, u_i\}$ and open API description text $s = \{s_1, s_2, s_3, \ldots, s_i\}$, where i is the size of the corpus vocabulary, and then used cosine similarity to calculate similarity:

$$\cos(\vec{q}, \vec{s}) = \frac{\vec{q} \cdot \vec{s}}{\|\vec{q}\|\|\vec{s}\|} = \frac{\sum_{i=1}^{v} \vec{q} \cdot \vec{s}}{\sqrt{\sum_{i=1}^{v} \vec{q}_i^{\,2} \sum_{i=1}^{v} \vec{s}_i^{\,2}}}. \tag{20}$$

(2) Open API discovery approach based on LDA (LDAOAD): LDA is a topic model, which can give the topic of each document in the document set as a probability distribution, so we used LDA to extract

MIM																					Y
Active open API				Target Mashup				Similar open API				Similar Mashup				Co-occurrence				Popularity	Score
1	0	0	…	1	0	0	…	0	0.6	0.5	…	0	0.3	0.7	…	0	0.5	0.5	…	12	1
0	1	0	…	1	0	0	…	0.4	0	0.7	…	0	0.3	0.7	…	0.5	0	1	…	3	0
0	0	1	…	0	0	1	…	0.5	0.8	0	…	0.3	0.4	0	…	0.5	1	0	…	5	1
1	0	0	…	0	0	1	…	0	0.6	0.3	…	0.3	0.4	0	…	0	0.5	0.5	…	7	0
0	1	0	…	0	1	0	…	0.6	0	0.7	…	0.5	0	0.6	…	0.5	0	1	…	1	1
0	0	1	…	0	1	0	…	0.5	0.7	0	…	0.5	0	0.6	…	0.5	1	0	…	8	1

Figure 4: The example of FM model input and output.

Table 3: User story extraction examples. According to Section 3.1, we extract open API properties on PWeb. Table 4 shows specific examples of open API properties.

No.	User story	Open API category	Requirements components
S1	As an editor, I want to download and edit pictures on the website	Photo	<editor, null>, <{download, edit},picture, null, null>, <null, null, website, null>
S2	As a producer, I want to search and upload music online	Music	<producer, null>, <{search, upload},music, null, null>, <null, null, null, online>
S3	As an editor, I want to download and upload videos from the app	Video	<editor, null>, <{download, upload},video, null, null>, <null, null, app, null>
S4	As a fan, I want to find music so that I can listen to music online	Music	<fan, null>, <find, music, null, null>, <listen, music, null, online>
S5	As a traveler, I want to find routes and hotel online	Travel	<traveler, null>, <find, {route, hotel}, null, null >, <null, null, null, online>

Table 4: Examples of open API properties extraction.

Name	Category	Open APIs properties
PicMonkey	Photos	<{maker}, null> <{upload, edit, save, contact}, {image, provider}> <{documentation,talk@picmonkey.com}, null}>
Google maps	Mapping	<map, null> <{utilize, access, embedding}, {language, localization, api, developer, geocoding, service, intranet}> <{service, customer, connection}, null>
Google-AdSense	Advertising	<{developer, blog}, null> <{create, generate, choose, generate, share, sense},{content, report, revenue, program, site}> <{account, snippet, filter}, null>

the subject distribution vector of user story u and open API description text s and then used enhanced cosine similarity to calculate similarity. The formula is as follows:

$$\mathrm{Sim}(a,u) = \frac{\sum_{i\in I}\left(r_{a,i} - \overline{r}_a\right)\left(r_{u,i} - \overline{r}_u\right)}{\sqrt{\sum_{i\in I}\left(r_{a,i} - \overline{r}_a\right)^2}\sqrt{\sum_{i\in I}\left(r_{u,i} - \overline{r}_u\right)^2}}. \quad (21)$$

Figure 6 shows that our approach is significantly better than the VSMOAD and LDAOAD approach, but the precision between TOP-20 and TOP-25 is significantly reduced. Because Stanford Parse cannot fully extract the open API demand components in some scenarios, open API function is represented by (1) noun phrases, such as "video upload," and (2) sentences with grammatical errors or missing structural components.

4.3. The Experimental Analysis for Open API Recommendation

4.3.1. Metric Selection. We adopt the precision, recall, and F-measure to evaluate the efficiency of the MIM-based open API approach:

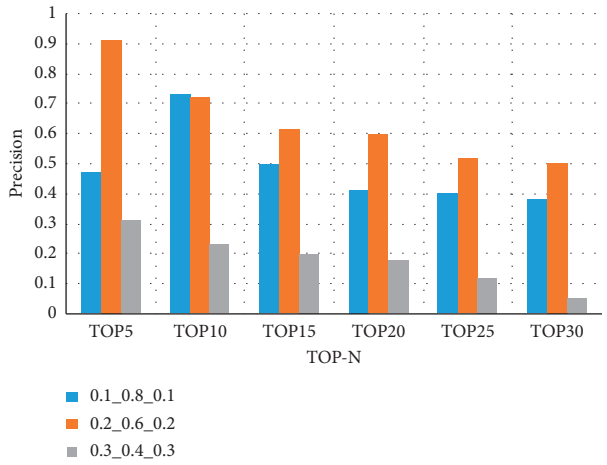

Figure 5: The result of selecting different parameters.

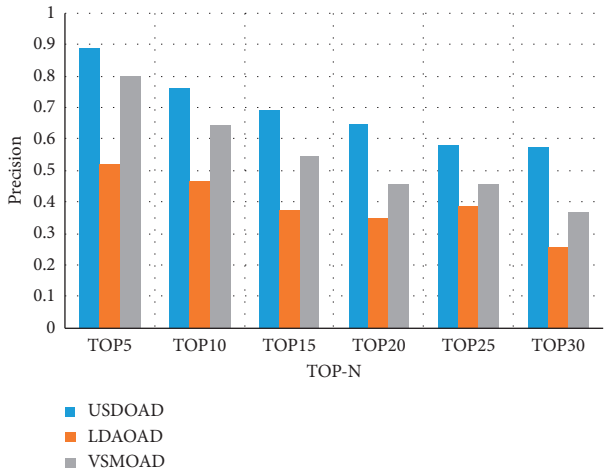

Figure 6: The precision results of comparing USDOAD, LDAOAD, and VSMOAD.

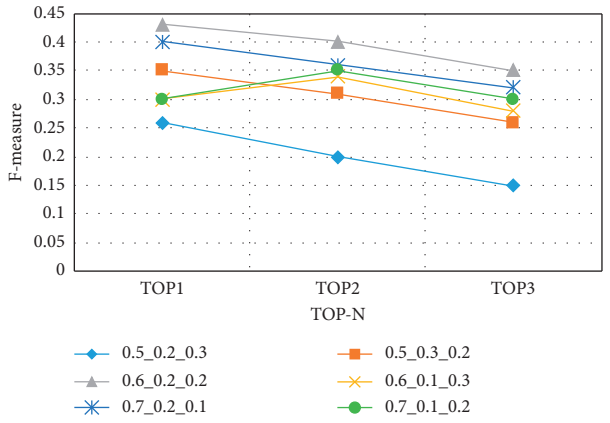

Figure 7: The *F*-measure value corresponding to each parameter.

$$P_{\text{recall}} = \frac{\left|R(A_i) \cap RM(A_i)\right|}{RM(A_i)},$$
$$P_{\text{precision}} \frac{\left|R(A_i) \cap RM(A_i)\right|}{R(A_i)}. \tag{22}$$

In the above two formulas, $R(A_i)$ represents the open API actually invoked by the target Mashup and $RM(A_i)$ represents the recommended open API from our approach.

F-measure is the unified average of recall rate and accuracy:

$$F\_\text{measure} = \frac{2 \times P_{\text{recall}} \times P_{\text{precision}}}{P_{\text{recall}} + P_{\text{precision}}}. \tag{23}$$

The relationship between the three metrics and the performance of the recommended algorithm is roughly positively correlated. The larger the recall, precision, and *F*-measure are, the better the performance of the recommended approach is; otherwise it has poor performance.

4.3.2. Parameter Selection. The similarity calculation formula for the open API-recommended algorithm invoked Mashup that is proposed in this paper contains three parameters: u, v, and w, which correspond to the function of Mashup, the application scenario and the similarity calculation of the invoked open API, and $u + v + w = 1$. They directly affect the construction of MIM, which indirectly influences the effect of FM model. Figure 7 shows the effect of the values of the five groups u, v, and w on the recommended results. It can be seen from Figure 7 that when the values of u, v, and w are 0.6, 0.2, and 0.2, the recommended effect is higher than other groups, so our parameters are configured as $u = 0.6$, $v = 0.2$, $w = 0.2$.

We choose the best values of TOP-A open APIs and TOP-M Mashups similar to achieve the best recommended results, as shown in Figures 8 and 9, respectively; when the number of recommended open APIs is 1, 2, and 3, the values of TOP-A and TOP-M (A is from 5 to 30 and M is from 5 to 30) affect the recommendation result. The experimental results show that when TOP-A remains unchanged and TOP-M = 20, F-measure is the best; when TOP-M remains unchanged and TOP-A = 10, the F-measure of MIM-based reaches its peak value. It turns out that selecting the appropriate TOP-A value and TOP-M value for the Mashup creation in the open API recommendation is very important.

4.3.3. Comparative Experiment. We compared MIM-based open API recommendation approach with other three recommendation approaches, which are TF-IDF [40], E-LDA [41], and LDA-FM [31].

(1) TF-IDF: This approach starts from the degree of similarity between the active open API description document and the target Mashup description

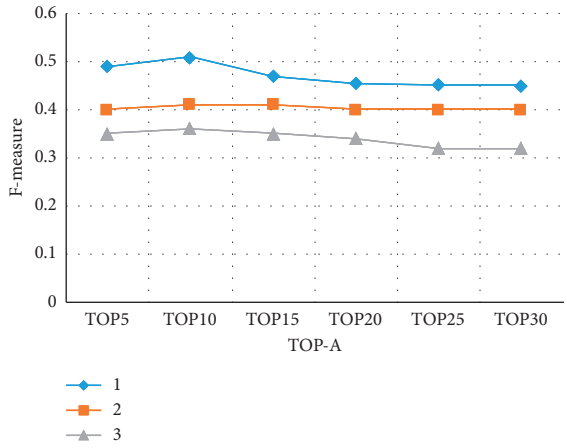

Figure 8: TOP-A's F-measure values.

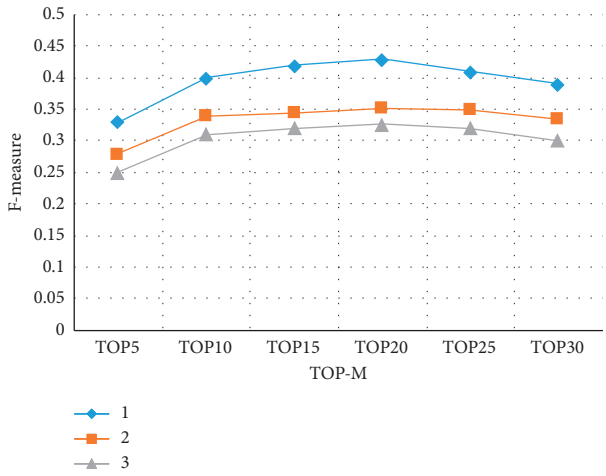

Figure 9: TOP-M's F-measure values.

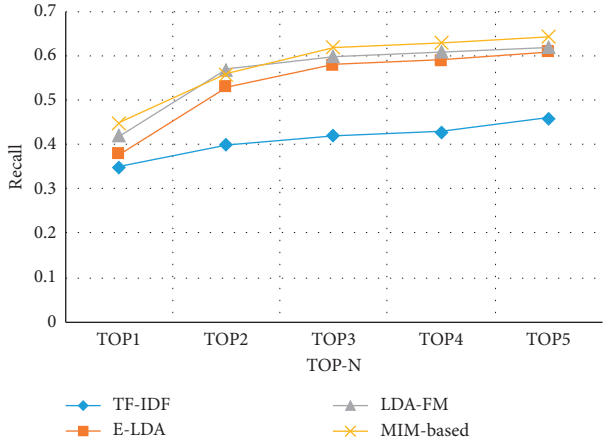

Figure 10: Results of the four approaches on recall.

document, and the score is calculated in conjunction with the open API popularity. First, we used the TF-IDF to calculate the word vector between the open API and the Mashup. Then, using the similarity of the word vector, we measured the similarity between the active open API and the target Mashup. Finally, we combined the similarity with the open API popularity to obtain the final open API recommendation score.

(2) E-LDA: This approach first calculates the topic vector of target Mashup and open API with LDA, then calculates their similarity, and then integrates the open API popularity for recommendation.

(3) LDA-FM: First, it extracts the topic distribution of the target Mashup and the active open API through the LDA model. The topic information is trained with FM to calculate the probability distribution of the open API. Similarly, we can get the similarity between target Mashup and active open API. In addition, it also takes the co-occurrence and popularity into account.

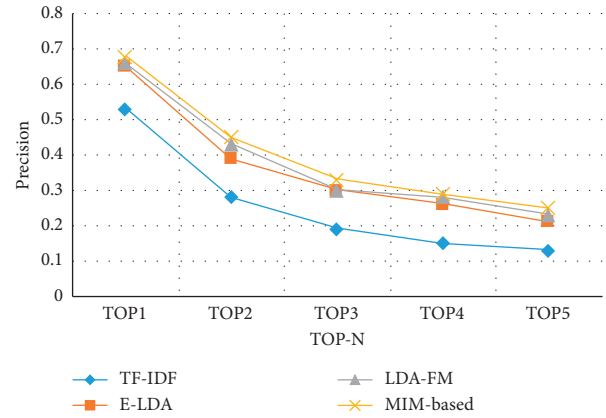

Figure 11: Results of the four approaches on precision.

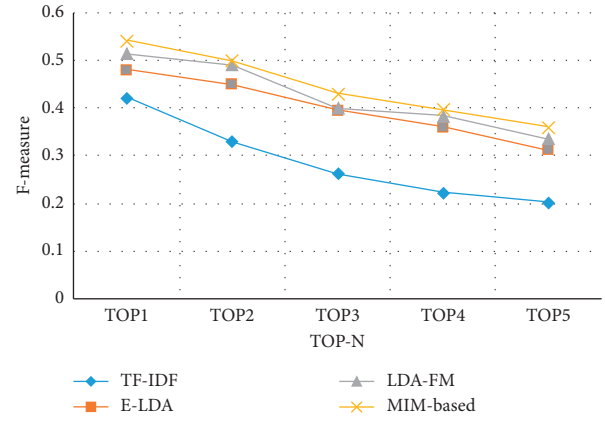

Figure 12: Results of the four approaches on *F*-measure.

The evaluation result is calculated in terms of the recall, precision, and F-measure [42]. The comparison shows that our approach has the highest accuracy in all the three metrics.

In Figure 10, our approach is better on recall than the other three approaches, and recall increases as N increases. In Figure 11, although the precision decreases as N increases,

our approach is still the best. As shown in Figure 12, the average *F*-measure value of the MIM-based approach is 2.21% higher than LDA-FM, 4.60% higher than E-LDA, and 15.81% higher than TF. In all cases, TF-IDF has the worst performance. TF-IDF just uses the frequency of word occurrences to vectorize words, regardless of the underlying semantic relevance behind them. MIM-based approach, LDA-FM, and E-LDA reveal the semantic relevance of open APIs and Mashup description documents, so they can calculate their similarities with higher accuracy.

5. Conclusions and Future Work

In order to enable Mashup developers to select the most suitable open API from a large set of open APIs in a rapid agile development mode, we propose a Hybrid Open API Selection Approach for Mashup development (HyOASAM). HyOASAM is composed of two basic approaches: a user-story-driven open API discovery approach and a MIM-based open API recommendation approach. Through HyOASAM, Mashup developers can (1) use user stories to describe open API requirements clearly and quickly, and (2) dynamically get a list of open APIs that match the requirements and select the open APIs they want. It can be seen through experiments that HyOASAM has improved in precision and recall. In the future we will consider employing Word Embedding and Attention Model into NLP techniques, so that the semantic relationship between words can be fully extracted.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

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

Acknowledgments

The authors wish to thank all the participants in the evaluation process for their help. This work was supported by the Zhejiang Provincial Natural Science Foundation of China (Grant no. LY19F020003) and the National Natural Science Foundation of China (Grant no. 61672459).

References

[1] W. Pan, B. Song, K. Li, and K. Zhang, "Identifying key classes in object-oriented software using generalizedk-core decomposition," *Future Generation Computer Systems*, vol. 81, pp. 188–202, 2018.

[2] W. Pan, B. Li, J. Liu, Y. Ma, and B. Hu, "Analyzing the structure of Java software systems by weightedK-core decomposition," *Future Generation Computer Systems*, vol. 83, pp. 431–444, 2018.

[3] W. Pan, H. Ming, C. Chang, Z. Yang, and D. K. Kim, "ElementRank: ranking java software classes and packages using a multilayer complex network-based approach," *IEEE Transactions on Software Engineering*, 2019.

[4] W. Tan, Y. Fan, A. Ghoneim, M. A. Hossain, and S. Dustdar, "From the service-oriented architecture to the web API economy," *IEEE Internet Computing*, vol. 20, no. 4, pp. 64–68, 2016.

[5] Z. Li, H. Zhang, and L. O'Brien, "Facing service-oriented system engineering challenges: an organizational perspective," in *Proceedings of the 2010 IEEE International Conference on IEEE Service-Oriented Computing and Applications (SOCA)*, pp. 1–4, Perth, Australia, December 2010.

[6] X. Liu, Y. Hui, W. Sun, and H. Liang, "Towards service composition based on mashup," in *Proceedings of 2007 IEEE Congress on Services (Services' 07)*, pp. 332–339, IEEE, Salt Lake City, UT, USA, Salt Lake City, UT, USA, July 2007.

[7] S. Aghaee and C. Pautasso, "Mashup development with HTML5," in *Proceedings of the 3rd and 4th International Workshop on open APIs and services Mashups*, p. 10, ACM, Ayia Napa, Cyprus, December2010.

[8] P. Massimo and T. Erl, *SOA Design Patterns (Paperback)*, Pearson Education, London, UK, 2008.

[9] F. Chen, M. Li, H. Wu, and L. Xie, "Web service discovery among large service pools utilising semantic similarity and clustering," *Enterprise Information Systems*, vol. 11, no. 3, pp. 452–469, 2017.

[10] M. Aznag, M. Quafafou, and Z. Jarir, "Leveraging formal concept analysis with topic correlation for service clustering and discovery," in *Proceedings of 2014 IEEE International Conference on Services (ICWS'14)*, pp. 153–160, IEEE, Anchorage, AK, USA, Anchorage, AK, USA, June-July 2014.

[11] M. Aznag, M. Quafafou, E. M. Rochd, and Z. Jarir, "Probabilistic topic models for web services clustering and discovery," *Service-Oriented and Cloud Computing*, pp. 19–33, Springer, Berlin, Heidelberg, 2013.

[12] L. Yao, Q. Z. Sheng, A. H. H. Ngu, J. Yu, and A. Segev, "Unified collaborative and content-based web service recommendation," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 453–466, 2015.

[13] L. Yao, Q. Z. Sheng, A. Segev, and J. Yu, "Recommending web services via combining collaborative filtering with content-based features," in *Proceedings of 2013 IEEE 20th International Conference on Services (ICWS'13)*, pp. 42–49, IEEE, Santa Clara, CA, USA, June-July 2013.

[14] J. Li, J. Wang, Q. Sun, and A. Zhou, "Temporal influences-aware collaborative filtering for QoS-based service recommendation," in *Proceedings of 2017 IEEE International Conference on Services Computing (SCC'17)*, pp. 471–474, IEEE, Honolulu, HI, USA, June 2017.

[15] T. Liang, L. Chen, J. Wu, H. Dong, and A. Bouguettaya, "Meta-path based service recommendation in heterogeneous information networks," in *Service-Oriented Computing*, pp. 371–386, Springer, Cham, Switzerland, 2016.

[16] Y. Wang, T. Wang, and J. Sun, "PASER: a pattern-based approach to service requirements analysis," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 4, pp. 547–576, 2019.

[17] Y. Wang and L. Zhao, "Eliciting user requirements for e-collaboration systems: a proposal for a multi-perspective modeling approach," *Requirements Engineering*, vol. 24, no. 2, pp. 205–229, 2019.

[18] M. Crasso, A. Zunino, and M. Campo, "A survey of approaches to web service discovery in service-oriented architectures," *Journal of Database Management*, vol. 22, no. 1, pp. 102–132, 2011.

[19] C. Mateos, J. M. Rodriguez, and A. Zunino, "A tool to improve code-first Web services discoverability through text mining techniques," *Software: Practice and Experience*, vol. 45, no. 7, pp. 925–948, 2015.

[20] A. V. Paliwal, B. Shafiq, J. Vaidya, H. Hui Xiong, and N. Adam, "Semantics-based automated service discovery," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 260–275, 2012.

[21] K. Elgazzar, A. E. Hassan, and P. Martin, "Clustering wsdl documents to bootstrap the discovery of web services," in *Proceedings of 2010 IEEE International Conference on Services (ICWS'10)*, pp. 147–154, IEEE, Miami, FL, USA, July 2010.

[22] C.-B. Ke, Z.-Q. Huang, L.-Y. Liu, and Z.-N. Cao, "Research on constraint-oriented web service discovery," *Journal of Software*, vol. 23, no. 10, pp. 2665–2678, 2012.

[23] H. Gao, S. Wang, L. Sun, and F. Nian, "Hierarchical clustering based web service discovery," in *Service Science and Knowledge Innovation*, pp. 281–291, Springer, Berlin, Germay, 2014.

[24] M. Klusch, *Semantic Service Coordination CASCOM: Intelligent Service Coordination in the Semantic Web*, Springer, Berlin, Germay, 2008.

[25] D. Wei, T. Wang, J. Wang, and A. Bernstein, "SAWSDL-iMatcher: a customizable and effective Semantic web service matchmaker," *Journal of Web Semantics*, vol. 9, no. 4, pp. 402–417, 2011.

[26] B. Cao, J. Liu, Y. Wen, H. Li, Q. Xiao, and J. Chen, "QoS-aware service recommendation based on relational topic model and factorization machines for IoT Mashup applications," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 177–189, 2019.

[27] Z. Zheng, H. Ma, M. R. Lyu, and L. King, "QoS-aware web service recommendation by collaborative filtering," *IEEE Transactions on Services computing*, vol. 4, no. 2, pp. 140–152, 2010.

[28] G. Huang, Y. Ma, X. Liu, Y. Lou, and X. Lu M. B. Blake, "Model-based automated navigation and composition of complex service Mashups," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 494–506, 2014.

[29] W. Xu, J. Cao, L. Hu, J. Wang, and M. Li, "A social-aware service recommendation approach for Mashup creation," in *Proceedings of the 2013 IEEE 20th International Conference on Web Services (ICWS'13)*, pp. 107–114, IEEE, Santa Clara, CA, USA, June-July 2013.

[30] W. Gao, L. Chen, J. Wu, and H. Gao, "Manifold-learning based api recommendation for Mashup creation," in *Proceedings of the 2015 IEEE International Conference on web Services (ICWS'15)*, IEEE, New York, NY, USA, pp. 432–439, June-July 2015.

[31] H. Li, J. Liu, B. Cao, and M. Shi, "Topic-adaptive open API recommendation method via integrating multidimensional information," *Journal of Software*, vol. 11, p. 10, 2018.

[32] B. Xia, Y. Fan, W. Tan, K. Huang, J. Zhang, and C. Wu, "Category-aware API clustering and distributed recommendation for automatic Mashup creation," *IEEE Transactions on Services Computing (TSC'14)*, vol. 8, no. 5, pp. 674–687, 2014.

[33] Y. Meng, A. Rumshisky, and A. Romanov, "Temporal information extraction for question answering using syntactic dependencies in an lstm-based architecture," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, (EMNLP'2017)*, pp. 9–11, Copenhagen, Denmark, September 2017.

[34] Z. Li, J. Wang, and N. Zhang, "A topic-oriented clustering approach for domain service," *Journal of ComPuter Resraech and Develpment*, vol. 51, no. 2, pp. 408–419, 2014.

[35] M. C. D. Marneffe and C. D. Manning, "The Stanford typed dependencies representation," in *Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation, Coling 2008*, Association for Computational Linguistics, Manchester, UK, pp. 1–8, August 2008.

[36] I. Lizarralde, J. M. Rodriguez, C. Mateos, and A. Zunino, "Word embeddings for improving REST services discoverability," in *Proccedings of the 2017 XLIII Latin American Computer Conference (CLEI)*, pp. 1–8, IEEE, Cordoba, Argentina, September 2017.

[37] W. Pan and C. Chai, "Measuring software stability based on complex networks in software," *Cluster Computing*, vol. 22, no. s2, pp. 2589–2598, 2019.

[38] C. Platzer and S. Dustdar, "A vector space search engine for web services," in *Proccedings of the Third European Conference on Services (ECOWS'05)*, p. 9, IEEE, Vaxjo, Sweden, November 2005.

[39] N. Zhang, J. Wang, K. He, and Z. Li, "An approach of service discovery based on service goal clustering," in *Proccedings of the 2016 IEEE International Conference on Services Computing (SCC)*, IEEE, San Francisco, CA, USA, pp. 114–121, June-July 2016.

[40] B. Cao, J. Liu, M. Tang, Z. Zheng, and G. Wang, "Mashup service recommendation based on usage history and service network," *International Journal of Web Services Research*, vol. 10, no. 4, pp. 82–101, 2013.

[41] C. Li, R. Zhang, J. Huai, and H. Sun, "A novel approach for API recommendation in Mashup development," in *Proccedings of the 2014 IEEE International Conference on Web services(ICWS)*, IEEE, Anchorage, AK, USA, pp. 289–296, June-July 2014.

[42] W. Pan, H. Jiang, H. Ming, C. Chai, B. Chen, and H. Li, "Characterizing software stability via change propagation simulation," *Complexity*, vol. 2019, Article ID 9414162, 17 pages, 2019.