# Introduction

With the popularity of service-oriented software development (SOSD), a large number of Web services have been released on the Internet and lots of developers have built their mashups (a new type of Web application) that can provide specific functionalities by integrating one or more Web services, which reduce the cost and workload of system development and increase the reusability and quality of software systems [1]. The rapid growth of the number of web services brings severe challenges to the management and reuse of services. Service recommendation, a technology that recommends appropriate component services for mashup developers during the development process, has emerged.

Existing approaches on service recommendation can be approximately classified into three types according to the explicit or implicit information leveraged in the recommendation process: content-based recommendation, collaborative filtering (CF) based recommendation and hybrid recommendation. Content-based recommendation approaches recommend services based on similarities between service descriptions and explicitly specified requirements.【】 CF-based algorithms mine implicit invocation preferences according to service invocation history.【】Hybrid approaches integrate explicitly requirements, implicit preferences, and other information of service usages such as co-invocation and popularity, to make recommendations. 【】.



##### (a) Static mashup development (b) Interactive mashup development

Fig. Typical mashup development scenarios

Existing service recommendation approaches are designed for mashup development in static scenarios, that is, they simultaneously recommend all candidate services for the mashup to be developed, which is indeed a single-round recommendation. However, such a single-round recommendation have some limitations. On one hand, user intentions cannot be correctly or comprehensively captured at the very beginning, and many interactive query extension expansion mechanisms [??] have thus been proposed. On the other hand, the recommendation result cannot be dynamically adjusted according to the instant decisions made by developers. We show the necessity of interactive recommendation using an example. Suppose a developer plans to build a mashup to help tourists prepare for their trip. Considering that tourists need navigation services when travelling, the recommendation list may include many mapping services, such as the Baidu Map, the Google Map, and so on. If the developer selects the Baidu Map from the list, the SRS should respond to this feedback and remove the Google Map in the next round of recommendations since it is unlikely for the developer to invoke another mapping service with the same function (particularly for these pay services). Moreover, the services that are closely related to the Baidu Map will have a higher recommendation priority.

Therefore, the interactive recommendation scenario has the potential to recommend more appropriate component services to meet developer requirements. As shown in Figure1.(b), suppose a developer wants to build a new mashup and enters his functional requirement into a service recommendation system (SRS). In the first stage, the SRS analyzes the requirement and generates a list of candidate services, from which the developer selects one or more component services as needed. If the recommendation result does not fully satisfy the requirement of the developer, the recommendation process gets into the second stage, where the SRS continues to generate a new service list according to the requirements and the selected services. This step loops until the mashup is completely set up. Such an online interactive scenario is different from the traditional static scenario, and it also has some special requirements for SRSs.

1. The entire developing process is indeed a session or a transaction. On the one hand, what a developer concerns in this process is how to realize his current short-term goal, that is, how to select some component services in terms of their functions and their relationship. On the other hand, this session is an integral whole and the next service is supposed to coordinate with the selected services to meet the developer requirements. Therefore, the SRS ought to consider the complex interaction among the mashup to be built, the selected services and the next service when making recommendations.
2. The session to build a new mashup is established and completed online. It can be divided into two stages: (1) when developers just initiate their sessions and have not selected any component services, SRS should make the first round recommendations according to their initial requirements; (2)afterwards, when developers have selected one or more component services, the SRS needs to make prompt response to the real-time feedbacks (i.e., the selected component services), that is, recharacterize developer requirements in a more accurate way and generate follow up recommendations in time. Indeed, the latter stage is the major difference between static and interactive recommendation, and the interaction between SRSs and developers is reflected in the exploitation of developer feedbacks.

Most of the existing service recommendation technologies cannot meet the above needs or work well in the above stages. Model-based CF methods, e.g., MF, NCF,LDA【】, cannot model new developers with no component services or make personalized recommendations in the first stage. In the second stage, even if developers have selected some services, they can hardly update their models in real time to obtain the latest representations of developers. In other words, they cannot dynamically generate follow up recommendations based on developer feedback. User-based CF and content-based methods only consider the relationship between the mashup and the next service and ignore the important role that the selected services play in the selection of next service, so they cannot mine developer feedbacks or update their recommendation list in the second stage. As the combination of content-based and CF-based algorithms, hybrid algorithms face the same problems as these two algorithms.

From the viewpoint of model realization, the first stage in the interactive mashup development scenario could be viewed as a special case of the second one, where the number of selected services is zero. Without loss of generality, the problem to be addressed in the paper is, **given developer requirements and some selected component services, how to recommend the next service for the developer to complete the mashup?** To address this issue, we propose a deep-learning based model that makes recommendations based on the interaction among the mashup, the selected services and the next service. Specifically, it employs Multi-Layer Perceptron (MLP) to capture the complex interaction and uses the **attention mechanism** to distinguish the influence of different selected services on the next service. Note that we can easily apply our model to the first stage by inactivating or masking its component designed for selected services. Our model is applicable to various kinds of information, for example, the content information and the invocation records between mashups and services. From each information, we first obtain the feature representations of a new mashup, the selected services and next service, then learn a specific interaction in this space and finally make a prediction. We can integrate the interactions in different spaces to make more accurate recommendation. Our model can serve online and make follow up recommendations in real time as long as some services are newly selected.

The main contributions of this work are:

1. We summarize a scenario of building a mashup in an online interactive session and analyze its requirements for the SRS that work in it.
2. We design an interaction-oriented model for online interactive session-based service recommendation. With the help of MLP and the attention mechanism, our model can recommend services that are compatible with the selected services and satisfy developer requirements when serving online.
3. We propose two modules that learn specific interaction using the content information and the invocation between mashups and services respectively. Combining these two modules, we obtain a hybrid model that integrates the interactions in different spaces and makes more accurate recommendation.
4. The experimental results on the real data set show that compared with the existing algorithms, our hybrid model has achieved the best results in various indicators.

# an interaction-oriented model FOR online conversational mashup development

## Problem Statement.

A service repository is represented as , where M is the set of mashups and S is the set of existing services. The invocation history between *M* and *S* can be regarded as implicit feedback data and converted into an invocation matrix *MS*, where the value at the *m*-th row and *s*-th column is:

.

Mashups and services have their respective attributes. Particularly, the content information (such as textual descriptions and tags) of a mashup or a service describes its requirements or functionalities. Each service also has other attributes, e.g., the provider.

Above all, the problem to be solved in this paper can be described as follows. Suppose a developer has provided his requirement (referred to as *MReq*) to build a new mashup and selected some component services (referred to as selected services and denoted as *SS*), how can the SRS recommend the next service to the developer based on the available information (i.e., *MReq*, the selected services, *MS*, the content and provider information in the repository)?

## An interaction-oriented Model.

In the interactive recommendation scenario, there exist complicated relations or interactions among MReq, selected services and the next service. On the one hand, the next service and the selected services may be replaceable or complementary to each other. On the other hand, the selected services and the next service work together to satisfy. The interactions help determine whether developers will select a candidate service as next service. Therefore, we design a deep learning-based model to capture the complex interactions and predict the probability of the developer selecting next. The K services with the highest probabilities are selected as the recommendation list.

Our model consists of three layers: a feature extraction layer, an interaction layer and an output layer.



### Feature extraction layer

In this layer, we input one type of information of a new mashup , the selected services *SS* and a service *s*, such as the content information or the invocation between mashups and services, and then get their feature representation, based on which we can learn their interactions in the same space. . Note that what we obtain in this layer is a representation of a mashup or a single service. We apply different feature extractors to the content information and the invocation and we will detail them in section 4.

### Interaction layer

This layer learns the interaction among *m*, SS, and s. Due to the complex relationship among them, the hasty assumptions and definitions on the interaction may not fit the reality. Considering that MLP can theoretically fit arbitrary function [], we employ the MLP to model the interaction.

An intuitive idea is that we first use multiple MLPs to learn the interactions among m, s and each selected service respectively, and then integrate these interactions by another MLP. But it consumes too much computing resources. Alternatively, we first compress the representations of all selected services into a lower-dimensional vector based on the attention mechanism. On this basis, it takes only one MLP to learn the complex interaction among m, s and SS.

#### Attention-based feature integration

To get an overall representation of all selected services, the simplest method is to directly **concatenate** their representations. However, the concatenation will increase the dimension of the final representation and reduce the efficiency of the model, especially when the size of selected services is large. What’s more, the next service is actually not sensitive to the order of the selected services. Obviously the simple concatenation result does not conform to this property.

Another popular solution is to preform average **pooling** or sum pooling on the representation of each service. Although generating a fixed-length representation, the pooling method has its disadvantage. Let us continue to analyze the case mentioned above, where a developer is building a mashup to help tourists for their trip. Suppose the developer has already selected several services including: a hotel reservation service, a weather forecast service, and a mapping service. Considering that tourists usually prepay online after booking a hotel, the hotel reservation service will have more influence on selecting an electronic payment service as the next service, compared with the weather forecast service and the mapping service. Therefore, each selected service has a different influence on the selection of the next service, which means that the representation of each service should be integrated with a distinct weight. However, the pooling method actually weight them equally.

According to the above analysis, we design an **attention-based** method to **integrate the representation of each selected service.** It **pays attention to the selected services that are more related to** (i.e., with similar functions to replace each other, or with different functions to complement each other) the **next service** and ignore unnecessary services.

We use the weighted sum of features of each selected service to represent all selected services:

,

where is the weight of a selected service . The physical meaning of is the degree of correlation between and *s*, in other words, the contribution of to the user's selection on *s*. is jointly determined by the features of and *s*. For two vectors, we measure their similarity by their element-wise multiplication and their difference by their element-wise subtraction. The results of these two operations can be seen as prior knowledge to help model the correlation between and *s*. We concatenate them with the features of and *s* and input the result into an MLP to automatically learn the correlation between and *s* and obtain a scalar score. The process can be described as:

,

where denotes element wise multiplication, denotes element-wise subtraction, denotes concatenation operation, and MLP represents all operations within an MLP.

Finally, we input this score into a softmax layer to calculate a final weight scalar **:**

Different from average pooling, we adaptively calculate the representation of selected service according to the correlation between each selected service and s. Therefore, using our method, different selected services have different contribution weights to the overall representation and the representation varies with the candidate service. This means that, compared with the average pooling method, our attention-based method can obtain a more adaptable representation and have a better expression ability.

#### MLP-based interaction learning

Up to now, we have obtained the representations of m, SS, and s, [i.e.](http://dict.youdao.com/w/i.e./#keyfrom=E2Ctranslation),. We first concatenate these representations and then utilize an MLP to capture the complex interaction among m, SS, and s. CNN and RNN are not suitable for this interaction learning task, because there are no local or sequential patterns in the concatenated representations. We select PReLU as activation function because it improves model fitting with nearly zero extra computational cost and little overfitting risk.【】the process can be denoted as

where  **is the learnt** complex interaction vector.

### Output layer

We feed the learnt interaction vector into a softmax layer. Its output represents the probability of m selecting s as the next service.