# 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 at the historic moment.

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 and implicit preferences to make recommendations. Furthermore, many hybrid recommendation approaches utilize other information of service usages, such as co-invocation and popularity, to improve recommendation performance 【】.



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

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-turn recommendation. In most practical development scenarios, developers gradually select component services with the interaction with the service recommendation system (SRS) .. The interactive scenario summarized from the real experience of mashup developers can be described as figure1.(b). Suppose a developer wants to build a new mashup and enters his functional requirement into SRS. in the first stage, SRS analyzes the requirement and generates a list of candidate services, from which the developer selects one or a few 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 SRS needs to continue to generate a new service list according to the requirement 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 SRS.

1. **The entire developing process is indeed a session or 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 next service is supposed to coordinate with the selected services to meet the developer’s requirements. Therefore, **SRS ought to consider the complex interaction among the mashup to be built, the selected services and next service when making recommendations.**

The session to build a new mashup is established and completed online. It can be divided into two stages: (1) when developers just start their sessions and have not selected any component services, SRS should make the first round recommendations according to their requirements; (2) afterwards, when developers have selected one or more component services, SRS needs to make prompt response to , model developers needs more accurately, and generate follow up recommendations Indeed, stage 2 is the main difference between static and interactive recommendation. And the interaction between SRS and developers is reflected in SRS’s exploitation of developers’ feedbacks. Most of the existing service recommendation technologies cannot meet the above needs or work well in the above stages. Model-based CF methods 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 user feedback. e.g., MF, NCF,LDA【】.. Although user-based CF and content-based methods can work in both stages, they only consider the relationship between the new mashup and next service, and ignore the important roles the selected services play in the selection of next service. As a combination of content-based and CF-based algorithms, hybrid algorithms face the same problems as these two algorithms.

In the online interactive mashup development scenario, the interaction between SRS and developers occurs in stage 2, where SRS captures and utilizes the feedback of developers to make follow up recommendation. Besides, the first stage is essentially a special case of the second one, where no service has been selected. Therefore, this paper mainly focuses on the second stage, that is, **given developer requirements and some selected component services in the current session, how can SRS recommend next service for the developer to complete the mashup?** To address this issue, we propose an interaction-oriented model that recommends next service based on developer’s real time feedback in the interactive scenario. It employs Multi-Layer Perceptron (MLP) to capture the **complex interaction** among the new mashup, the selected services and next service. Specifically, it 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 users’ developing 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 information 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 R=M∪S, 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 the provider attribute.

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 one or more component services (referred to as the selected services), how can SRS recommend 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) at this time?

## An interaction-oriented Model.

In the interactive recommendation scenario, the services that the developer has selected in the current session, as a kind of developer feedback, plays an important role in the recommendation of next service. Let us first analyze a case. Suppose a developer wants to build a mashup to help tourists prepare for their trip. Considering that tourists need navigation service when travelling and being ignorant of developer preferences for different map services, the recommended list may include many map services, such as Baidu Map, Google Map, etc. If the developer selects Baidu Map from the list, SRS should respond to this feedback and remove Google Map in the next round of recommended results. The reason is that the demand for map service has been met, and it is unlikely for the developer to invoke another map service with the same function (this rule is more applicable to paid services). At the same time, the recommended results may introduce some services related to Baidu Map.

Overall, there exists a complicated relationship or interaction among MReq, selected services and next service. On the one hand, next service and the selected services may be replaceable or complementary to each other. On the other hand, the selected services and next service work together to satisfy requirement . 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: the feature extraction layer, the interaction layer and the output layer.



### The feature extraction layer:

Many kinds of information is available in the repository, such as the content information and the invocation between mashups and services. In this layer, we input one kind of information of a new mashup , the selected services and a service s, and then get their feature representation in the same space using some strategies and feature extractors. Note that what we get here is the representation of a mashup or a single service.

### The interaction layer

This layer learns the interaction among m, the selected services, 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 the selected services.

#### Attention-based feature integration

To get an overall representation of the 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, next-item is actually not sensitive to the order of the selected services. But the concatenation result does not satisfy this property obviously.

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. Suppose the developer who wants to build a mashup to help tourists prepare for their trip has already selected a service to make hotel reservation, a weather forecast service, Baidu Map and an electronic payment service, Alipay. Considering that tourists need to prepay fee online after booking a hotel and Alipay has been selected, the developer may invoke WeChat Pay to complement Alipay next. Obviously, the only services that play a leading role in the selection of WeChat Pay are the hotel reservation service and Alipay. So we believe that each selected service has a different influence on the selection of next service. It means that the representation of each service should be integrated with a distinct weight. However, the pooling method actually weight them equally.

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

We use the weighted sum of feature of each selected service to represent all selected services,*SS*.

where is formally the weight of a selected service . The physical meaning of is the degree of correlation between and *s* or the contribution of to the user's selection on *s*. is jointly determined by the feature 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 feature of and *s* and input the result into an MLP to automatically learn the correlation between and *s* and obtains 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 the method of pooling, we adaptively calculates 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, the selected services, and s, **[i.e.](http://dict.youdao.com/w/i.e./" \l "keyfrom=E2Ctranslation)**,. We first concatenate these representations and then utilize MLP to capture the complex interaction among m, the selected services, 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. 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 and regard the output value on “1” as the probability of the service would be selected as next service.