Learning to Reuse User Inputs in Service Composition

Shaohua Wang, Ying Zou
Queen's University, Kingston, Ontario, Canada
e-mail: shaohua@cs.queensu.ca, ying.zou@queensu.ca

Joanna Ng, Tinny Ng
IBM CAS Research, Markham, Canada
e-mail: {jwng, tng}@ca.ibm.com

***Abstract*—Users visit web services and compose them to accomplish on-line tasks. Normally, users enter the same information into various web services to finish such tasks. However, repetitively typing the same information into services is unnecessary and decreases the service composition efficiency. In this paper, we propose a context-aware ranking approach to recommend previous user inputs into input parameters and save users from repetitive typing. We develop five different ranking features constructed from various types of information, such as user contexts. We adopt a learning-to-rank approach, a machine learning technology automatically constructing the ranking model, and integrate our ranking features into a state-of-the-art learning-to-rank framework. Our approach learns the information of interactions between input parameters and user inputs to reuse user inputs under different contexts. Through an empirical study on 960 real services, our approach outperforms two baseline approaches on ranking values to input parameters of composed services. Moreover, we observe that textual information affects the ranking most and the contextual information of location matters the most to ranking among various types of contextual data.**

***Keywords*-information reuse; service composition; learning-to-rank; input parameter value recommendation**

I. Introduction

To conduct daily on-line tasks, users have to visit web-sites and on-line services to compose services together repeatedly [1]. During the service composition, users are required to provide values to input parameters of services to invoke each service. For example, planning a holiday trip from Toronto to Paris can involve a composition of two services: *Priceline*[1] and *The Weather Network* (TWN)[2]. A user purchases an airplane ticket on *Priceline* and checks the weather status of Paris on TWN. The user needs provide the information, such as city name, for checking weather, the contact and credit card information for reserving an airplane ticket. Filling values into services can be tedious, especially when the number of services is high and the previously entered values are required by other or same services repetitively.

Recently, several approaches have been developed to aid users in reusing previous user inputs to fill in services. For example, Chrome Autofill [2] recommends a list of previous user inputs to users. Firefox Autofill [3] pre-fills web forms using previous user inputs. Araujo *et al.* [4] propose a concept-based approach for pre-filling values to web forms. Firmenich *et al.* [5] propose an approach to annotate web forms and store user information in a centralized space for web form filling. AbuJarour *et al.* [6] propose an automatic approach to sample services and assign values to input parameters of services. However, the existing approaches face at least one of the following challenges: First, the approaches are not designed for helping users in filling values to input parameters from different types of services. Second, the existing approaches perform poorly in ranking or pre-filling values to input parameters. Third, the approaches are not context-aware. For example, they cannot distinguish a phone number for receiving a shipping package and the other one for travel contact.

To address the aforementioned challenges, we propose a ranking-based approach, ranking a list of previous user inputs for an input parameter, to save a user from unnecessary data entry during a service composition. Our approach pre-fills an input parameter with the top ranked value of a list and recommends the rest of the values to a user if he or she is not satisfied with the pre-filled value. We propose five ranking features derived from various types of information which could affect the ranking of previous user inputs, such as user contexts and user's past activities. We adopt a learning-to-rank (LTR) approach [7][8], a supervised machine learning approach that automatically builds a ranking model from training data [9], and integrate our ranking features into a start-of-the-art ranking model named RankSVM [10] a pairwise learning-to-rank model. When a user input is entered into an input parameter, an interaction between the user input and the input parameter is established. Our approach analyzes and learns all the information of the past interactions between user inputs and input parameters to reuse user inputs for input parameter filling. We refer our approach as a learning-to-reuse approach. The major contributions of this paper are listed as follows:

- We propose a context-aware meta-data model for capturing and storing user inputs with contextual information, such as time. The stored contextual information of user inputs makes the context awareness of our approach possible.
- We propose a user model to describe different aspects of a user, such as user contexts. The model is used for

1. $http://www.priceline.com/$
2. $http://www.theweathernetwork.com$

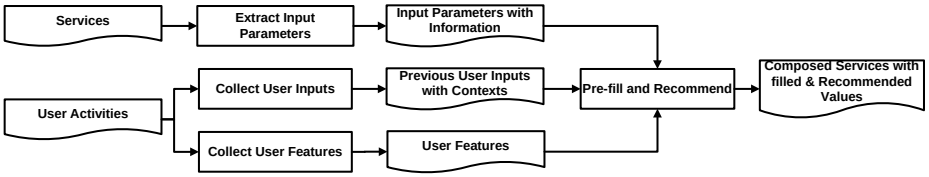

Figure 1: Overall steps for recommending the user inputs to input parameters during a service composition.

conceiving the ranking strategies for our approach.

- We propose a context-aware learning-to-rank model based approach to identify proper values for input parameters. Our approach exploits the information of a user(*e.g.*, user contexts), input parameters and user inputs for learning the past interactions between user inputs and input parameters to reuse user inputs for input parameter filling.
- We empirically test the effectiveness of our approach on real services. We compare our approach with two baseline approaches. Our approach outperforms the baseline ranking approaches. Our approach achieves a precision of 83% and a recall of 83% in pre-filling parameters. We observe that the textual similarity based feature affects the ranking most.

Paper organization. Section II presents the background of this study. Section III introduces our proposed approach. Section IV states the case studies. Section V summarizes the related literature. Section VI concludes the paper.

II. Background

Our paper is primarily focused on assisting end-uses in filling values into services during a service composition. A web service is a software module enabling inter-operation between machines over the web. In this paper, our approach handles the following three types of web services:

- *Web Services Description language (WSDL)*[3] *services* is an XML-based description language for describing the functionality of a web service.
- *RESTful services [11]* is proposed to simplify the development, deployment and invocation of web services. RESTful services use standard HTTP and permit various data formats. Web Application Description Language (WADL) [12] is an XML description of HTTP-based web applications.
- *Web Application services.* The users can also conduct various tasks through web applications (*e.g.*, web forms). A web application is viewed as a type of online web services. The WSDL and RESTful services are treated as the services with descriptions. The web forms of web applications are treated as the services without the formal descriptions.

A service has a set of input parameters intaking values from users. To invoke a service, every required input parameter should be filled with a proper value. A user input is a piece of information entered into services. The user inputs entered previously into services are an excellent source for discovering a proper value for an input parameter [4].

III. Overview of Our Approach

Figure 1 shows the steps of our approach. Our approach consists of three major steps:

1) *Collecting and storing user inputs with contextual information*. When a user enters a value to an input parameter, we collect all the information of the interaction between the value and the input parameter, such as the time of the interaction and the location where the user enters the value.
2) *Collecting and analyzing user information*. The user information describes different aspects of a user, such as user contexts. The user information can be the key factors for ranking an ideal list of user inputs for parameters. To exploit such information, we first capture and analyze such information.
3) *Ranking values for filling in operations of services*. When a user needs provide a value to an input parameter of a service during a service composition, a list of possible user inputs should be ranked to aid users in selecting the most suitable value to the input parameter. We propose a learning-to-rank based approach using five ranking features constructed from user features to rank the user inputs.

A. Collecting and Storing User Inputs from Users with Contextual Information

To help users fill in values to input parameters under different contexts, we need a mechanism to organize and store user inputs efficiently. We propose a context-aware meta-data model for storing and organizing user inputs.

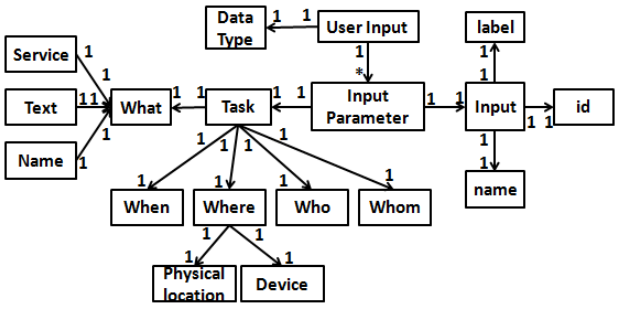

Figure 2: Description of context-aware meta-data model

1) Context-aware meta-data model of user input: Figure 2 shows the description of the proposed context-aware meta-data model. A user input is associated with two properties: *Type* and *Input Parameter*. The *Type* property records the data type of the user input. An *Input Parameter* property has two properties: *Task* and *Input* property.

3. http://www.w3.org/TR/wsdl

Task Property: A *Task* property has five sub-properties to store textual description and contextual information of a task where the input parameter is entered with a user input. The five sub-properties are listed as follows:

- ***What*** stores the information of what a task is. This property has three sub-properties:
  - ***Name*** records the name of a task, such as the name of an operation in WSDL, the resource name and its associated actions of a RESTful service, and a web label (*e.g.*, from a web form or HTML tag).
  - ***Text*** stores the description of a task. A task description is retrieved from the description of an operation in WSDL, the web page introducing a resource and its associated actions in a RESTful service, or textual information of a web form. The value of this property can be NULL.
  - ***Service*** records the name of a service where the task is performed. The value of this property can be the name of a WSDL, RESTful service, or the URL of a web form.
- ***When*** records the time when a task is performed by a user. The time can be obtained from the user's operating system.
- ***Where*** remembers the physical locations and computing devices of a user where he or she performs a task. The physical locations can be retrieved and calculated from IP or GPS (*e.g.*, Mobile devices).
- ***Who*** stores the identity of a user who performs the task. The way of obtaining the identity is dependent on the implementation choices. For example, the value of this property can be obtained from user profiles.
- ***Whom*** records the information whom a task is performed for. The value of ***Whom*** can be equal to the one of ***Who***. For example, a user buys a phone for himself or herself. The way of obtaining the social relation between ***Who*** and ***Whom*** is dependent on the implementation choices. For example, a relation can be retrieved from social networks, such as Facebook[4].

Input Property: An *Input* property stores the textual information of an input parameter, and has three sub-properties: *label*, *name*, and *id*. The textual information is mined from coding information. In web applications, *label*, *name*, and *id* are the attributes of the HTML DOM elements defining the input fields consuming user inputs. In the context of WSDL and RESTful services, we store the name of the input parameter in the *label* property.

For each property stored in the context-aware mete-data model, we conduct word normalization to identify meaningful words. We decompose any possible compound words (*e.g.*, FindCity) following the conventions used in programming languages. We use four rules to decompose words: case change (*e.g.*, FindCity), suffix containing a numeric number (*e.g.*, City1), underscore separator (*e.g.*, departure_city) and dash separator (*e.g.*, Find-city). We use WordNet[5] a lexical database to remove non-English words. We remove stop words using a pre-defined list of stop words, such as "the". Finally, we use porter stemmer [13] to reduce derived words to their stem, base, or root form.

2) Collecting user inputs and their contexts: Nowadays, electronic devices, such as mobile devices, are indispensable in people's life. A massive amount of user activities can be conducted using these electronic devices. For example, the users can conduct web activities or web service testing on PC or mobile devices. The users can shop a pair of shoes on Ebay[6] or register a course by filling in university registration web forms. The users can operate a simple RESTful web service testing using its URL through web browsers, or a SOAP-based WSDL service through SOAPUI framework[7]. The user inputs can be collected through such a massive amount of user activities.

We collect user inputs through users' web activities. We modify the tool used in [14] to collect user inputs. The original tool in [14] extends a web testing tool named Sahi [8] to monitor users' web activities and collect the user inputs. In this paper, we extend the original tool to include contextual information, such as time and location, of the interactions between user inputs and input parameters. We store the user inputs with their property information in the proposed context-aware meta-data model.

B. Collecting and Analyzing User Information

To aid users in filling values to input parameters under different circumstances, it is essential to understand the dynamicity of users' needs. We propose a user model to describe different aspects of a user. We analyze user information (*e.g.*, current user contexts and social relations) to discover the changes of users' needs. Our user model has five aspects are listed as follows:

- **User Profile**. A user profile contains a set of pieces of personal information, such as name, home address and credit card information. The number of entries in a user profile can be various based on different definitions or implementations. In this paper, we only include the basic personal information which is *user's real name, residency address, phone numbers, credit card information, date of birth, and email addresses*.
- **User Past Activities**. This aspect records all the past performed tasks by a user. Each activity entry stores the tasks, the services composed to accomplish the tasks, input parameters entered with values, and the entered user inputs.

4. https://www.facebook.com/
5. http://wordnet.princeton.edu/
6. http://www.ebay.com
7. http://www.soapui.org/
8. http://sahi.co.in/

- **User Social Relations**. This aspect stores the social relations among users. The relations, such as husband to wife, can be directly extracted from social network platforms. Knowing the relations among users could improve input parameter filling process. For example, sending a postcard to a co-worker is a common task. The co-worker's home address and phone number can be auto-filled into a service where the input parameters requiring shipping address.
- **User Contexts**. The definitions of user contexts are varied from different academic studies. We adopt the definition and taxonomy of user contexts in [15]. We discover four types of user contexts: time, location, identity and activity, useful for recommending values to input parameters. For example, a user may only check his or her account balance at home.
- **User's Current Typing**. Before a user finishes typing his or her intended value into an input parameter, the unfinished typing could be a perfect indicator for discovering the intended value. During the course of value typing, a user should be recommended with a set of ranked values based on the typing which are dynamically changing.

The aspects, **User Profile** and **User Past Activities**, are the main sources for providing values to input parameters. The aspect, **User Social Relations**, links the different users. More importantly, the aspects, **User Social Relations**, **User Contexts** and **User's Current Typing**, can be used for locating a suitable value for an input parameter.

C. Ranking and Recommending values to users for filling in operations of services

In this section, we first summarize four scenarios of how a user interacts with input parameters and how a user can be helped with the auto-filling techniques, when a user tries to supply values to input parameters of a service participating in a service composition. Then, we introduce our approach that ranks and recommends values to users.

1) Scenarios of user-parameter interactions: The four scenarios are listed as follows:

- *Scenario 1.* An input parameter is pre-filled with a value when no typing action is performed from a user.
- *Scenario 2.* A user is not satisfied with a pre-filled value. The user would remove the pre-filled value, then start typing a new value. Before the typing starts, a list of ranked values should be recommended to the user.
- *Scenario 3.* No value is selected by a user from the recommended list of values, then the user starts typing a new value. A list of ranked values should be recommended based on what the user is typing.
- *Scenario 4.* No previous user inputs are suitable for recommendation. The user has to type in a value.

Our proposed context-aware ranking approach helps users in *Scenario 1*, *Scenario 2* and *Scenario 3*. Especially, our approach can maximize the help for users in *Scenario 1* and *Scenario 2* which requiring users to type.

2) Context-aware ranking of user inputs: To help a user fill in an input parameter P, we recommend a list of ranked previous user inputs. All the user inputs are stored in the proposed context-aware meta-data model (Section III-A1). We propose a learning-to-rank model based approach incorporating the user information as features to recommend a personalized list of previous user inputs to users.

Model Formulation: A learning-to-rank (LTR) task has training and testing phrases. Given a set of queries $Q = \{q^1, q^2, \dots q^m\}$, where m is the number of queries. Each query q^i $(1 \leq i \leq m)$ is mapped to a list of documents $D^i = \{d_1^i, d_2^i, \dots, d_n^i\}$, where d_j^i denotes the j^{th} document. Each list of documents D^i is mapped to a list of relevance judgments $Y^i = \{y_1^i, y_2^i, \dots, y_n^i\}$, where y_j^i is the relevance judgment on document d_j^i with respect to query q^i. For each query-document pair (q^i, d_j^i), a feature vector is created. Features are defined as functions of a query-document pair. The learning task is to automatically learn a function $F(x)$ given a training data. The feature vectors are ranked according to $F(x)$, then the top K results are evaluated using their corresponding relevance judgments. Learning-to-rank approaches can be modified and developed into settings where no training data is available before deployment [16]. Compared with the conventional ranking models (*e.g.*, Bayesian Belief Networks [17] based ranking model), LTR models automatically learn and combine different factors affecting a ranking to suggest the ideal ranked list. We formulate the input parameter filling task into a LTR task. An input parameter and a previous user input are viewed as a query and a document respectively. The value of the relevance judgment on a user input I with respect to an input parameter P can only be 0 or 1, where 1 means I is perfect for filling in P and 0 means the opposite. We adopt a state-of-the-art LTR model, RankSVM [10] a pairwise LTR approach, to rank the possible user inputs.

Ranking Features: We propose five ranking features derived from our proposed user model.

Ranking Feature 1 - Textual Similarity: A previous user input sharing the most textual similarity value with the input parameter could be the right value and selected by the user. This ranking feature is based on the textual similarity between a user input and an input parameter.

Given a user input I and an input parameter P, we traverse all the *input parameter* properties (ipps) of the user input I. For each ipp, we calculate the similarity between the *input* property of the ipp and the textual information of the input parameter P. The *input* property of an ipp (denoted as $I(inp\{ipp\})$) and the textual information of the input parameter P (denoted as $P(text)$) can have more than one word (*e.g.*, "book flight"). We formulate $I(inp\{ipp\})$ and $P(text)$) into a bag of words, $bag = \{W_1, W_2, \dots, W_n\}$, where n is the number of words in a bag. We use WordNet

to calculate the semantic similarity between two bags bag_i and bag_j ($i \neq j$) in the following cases:

Case 1. We first identify common words (*i.e.*, two words are identical or synonymous) of two bags. If all of the words from two bags are same, then bag_i=bag_j. If bag_i has more words than bag_j does, and all of the words in bag_j are also in bag_i, we use Equation (1) to calculate the similarity between two bags.

$$sim(bag_i, bag_j) = \frac{|bag_i \cap bag_j|}{|bag_j|} \quad (1)$$

where $sim(bag_i, bag_j)$ denotes the semantic similarity value between bag_i and bag_j, $bag_j \subset bag_i$.

Case 2. If bag_j and bag_i are not contained by each other, we count the number of pairs of common words in two bags. Second, we calculate the semantic similarity between every pair of words W_a^i ($W_a^i \subset bag_i$) and W_b^j ($W_b^j \subset bag_j$) of two bags after removing the common words from calculation. Then, we add up all of the similarity values of pairs of words, denoted as Sim_{sum} = $\sum_{W_a^i \subset bag_i} \sum_{W_b^j \subset bag_j}$ sim(W_a^i, W_b^j), where *sim(W_a^i, W_b^j) denotes the similarity value between two words; $W_a^i \neq W_b^j$*. Last, we use Equation (2) to calculate the similarity between two bags. The numerator of the equation is to calculate the similarity between each pair of words of two bags; the denominator is the total number of times of calculating the similarity between words.

$$sim(bag_i, bag_j) = \frac{|bag_i \bigcap bag_j| + Sim_{sum}}{|bag_i \bigcap bag_j| + |pairs\ of\ nonidentical\ words|} \quad (2)$$

where $sim(bag_i, bag_j)$ denotes the semantic similarity value between bag_i and bag_j. $|bag_i \bigcap bag_j|$ states the number of pairs of common words of bag_i and bag_j. Sim_{sum} is the sum of the similar value of every pair of words excluding the common words from bag_i and bag_j.

In total, we obtain a set of textual similarity values between the *input parameter* properties of user inputs and current input parameter P. We choose the highest value as the score for the proposed ranking feature.

Ranking Feature 2 - Task Similarity: If one of the tasks associated with a user input can match the current task being performed by a user, the user input could be selected by the user. A task is an operation in WSDL services, or a resource and its associated action if applicable (*e.g.*, update) or a web form of a web application. We propose one ranking feature based on the similarity between previous tasks and the current task.

Given a set of *input parameter* properties (ipps) of a user input I and a current task, we calculate the textual similarity between the *task* property of each ipp (denoted as $tp\{ipp\}$) and the current task. Each $tp\{ipp\}$ has a *what* property describing the information of a task; each *what* property has three sub-properties: name (denoted as $n^{(what\{tp\{ipp\}\})}$) storing task name, text (denoted as $t^{(what\{tp\{ipp\}\})}$) storing task description and service (denoted as $s^{(what\{tp\{ipp\}\})}$) storing the name of a service where the task is performed as proposed in Section III-A1. We extract the name, description and service name of current task, denoted as n^{ct}, t^{ct} and s^{ct} respectively. We adopt the approach of calculating similarity between properties in ***Ranking Feature 1*** to calculate the following three similarities: $sim(n^{(what\{tp\{ipp\}\})}, n^{ct})$, $sim(t^{(what\{tp\{ipp\}\})}, t^{ct})$ and $sim(s^{(what\{tp\{ipp\}\})}, s^{ct})$. We use Equation 3 to calculate the similarity between a previously stored task (pt) and the current task (denoted as ct).

$$sim(pk, ct) = a * sim(n^{(what\{tp\{ipp\}\})}, n^{ct}) + b * sim(t^{(what\{tp\{ipp\}\})}, t^{ct}) + c * sim(s^{(what\{tp\{ipp\}\})}, s^{ct}) \quad (3)$$

where a, b, c are weights for similarity calculation. a, b and c have same value (i.e., $1/3$) derived empirically.

In total, we obtain a set of similarity values. We choose the highest value as the score for the proposed ranking feature.

Ranking Feature 3 - Contextual Similarity: The contexts associated with the interactions between user inputs and input parameters are important for recommendation. Given an input parameter and current user contexts, we identify the user inputs whose contexts matches current user contexts. A context can be a location or device where a user enters a value to a service. In the context-aware model of user input (Section III-A1), every user input is associated with time, location (*i.e.*, physical location and computing device), who performed a task and for whom. Each input parameter P to user input I interaction can be modeled into a tuple $T = \{P, C^{ip-ui}, I\}$, $C^{ip-ui} = \{c_i, c_2, \ldots, c_t\}$, where C^{ip-ui} is a set of contexts recorded during interactions, t is the number of contexts. We conduct context abstraction process [18], critical for context-aware recommendation, to transform the numerical values into categorical ones. We skip the detail introduction of the process due to page limit.

Given a set of tuples of P to I interactions and a set of current user contexts $C^{ct} = \{c_1^{ct}, c_2^{ct}, \ldots, c_m^{ct}\}$ (m is the number of contexts, $m \leq t$), we use the following steps to calculate the possibility of using a user input under current user contexts: First, we calculate the similarity between C^{ct} and the set of contexts (denoted as C^{ip-ui}) of every tuple, denoted as $sim(C^{ct}, C^{ip-ui})$. We turn the C^{ct} and all of the sets of contexts of tuples into bag of words. We use term frequency and inverse document frequency (tf-idf) [19] to weight each word and turn C^{ct} and every C^{ip-ui} into a vector. Second, we use cosine similarity algorithm [20] to calculate the similarity value of $sim(C^{ct}, C^{ip-ui})$. Our model is extensible with more types of contexts which could potentially affect ranking.

Ranking Feature 4 - Frequency: The ranking feature is built on the number of times a user input is provided to an input parameter. We mine the frequency of a user input to

an input parameter interaction from the context-aware metadata model of user input.

Ranking Feature 5 - Current Typing: The ranking feature is based on what a user is currently typing into an input parameter. To rank user inputs dynamically, the user inputs should contain what he or she types in as a prefix.

The five ranking features are designed for ranking previous user inputs in ***Scenario 1, 2 and 3 of user-parameter interactions***. The feature 5 (*i.e.*, Current Typing) is specialized for ***Scenario 3 of user-parameter interactions***.

IV. Case Study

We introduce our dataset and two research questions. For each question, we present the motivation of the question, the analysis approach and our findings.

A. Case Study Setup

We conducted our study on three types of services: WSDL services, RESTful services, and web applications (*i.e.*, web forms). The RESTful services are described in web pages.

Table I: Our collection of public services.

Domain	# of WSDL	# of REST	# of Web forms
Travel	215	30	50
E-commerce	190	30	50
Finance	135	30	50
Entertainment	100	30	50

We collect 640 public available WSDL files. We use programmableWeb[9] to collect 120 URLs of RESTful services and download the web pages containing the APIs. We manually collect the information related to RESTful services, such as the description of resources and input parameters. We use Google to search for websites to download web forms. We choose websites listed on the top of the Google results. In total we download 200 Web forms. The total 960 services fall into 4 domains: Travel (*e.g.*, book flights), E-commerce (*e.g.*, buy shoes), Finance (*e.g.*, check a stock price) and Entertainment (*e.g.*, search TV shows). Table I provides a summary of the dataset used in our case study. In total, we collect 11127, 792, 1024 input parameters from WSDL, Restful, and Web application services respectively.

User input collection: To evaluate the effectiveness of our learning-to-reuse approach, we collect a set of recorded user inputs through our input collector (Section III-A). We recruit 6 subjects who are graduate students and typically spend 8-10 hours on-line per day to use the input collector tool to track their inputs on web forms. When the subjects conduct a task, we require them to enter whom the task is performed for. The social relation between subject and whom his or her task targets for can be retrieved from different social platforms. Since all of the collected user inputs are stored in context-aware model of user inputs (Section III-A1), the relations between input parameters and user inputs (denoted as ip-ui interactions) are established. The ranking feature 5 (*i.e.*, current typing) requires a set of incomplete user inputs as the user current typing. The longer an incomplete input is, the easier identifying a suitable user input is, because the incomplete input is the pre-fix of the suitable user input. We extract the first character of a user input as the user current typing. In total, we collect 12584 cases of ip-ui interactions. For each case, we calculate the feature values for all the five features. We refer this dataset as *auto-data*.

Manually-labeled training dataset: To test our approach on the collected services, we need create the training dataset in the following steps. First, we sample input parameters of each type service with the confidence level 95% [21] so that we can manually judge the relevance between a user input and an input parameter. Second, for each input parameter, we retrieve a list of user inputs whose textual information matches the textual information of the input parameter. Third, we judge the relevance of a user input to the input parameter. There are three degrees of relevance: 1 (*i.e.*, best value for the input parameter), 0 (*i.e.*, not suitable for the input parameter) and not sure. We calculate the feature values for all the five ranking features. We use the same approach used in *auto-data* to build values for feature 5. We refer the dataset as *manually-labeled-data*.

B. Research Questions

We conduct experiments to measure the effectiveness of our approach and answer the following research questions.

RQ 1. Is the proposed ranking approach effective in ranking user inputs?

Motivation. Ranking user inputs to input parameters of services is an essential step to save users from repetitive typing. Ideally, the top-1 value of a ranked list is for input parameter pre-filling and the rest is for recommendation if a user is not satisfied with the pre-filled value. The ranking can be adjusted and adapted to the dynamic changes of user contexts. In this question, we evaluate the effectiveness of our learning-to-reuse approach in ranking inputs.

Approach. We build two baseline approaches. The first baseline ranks user inputs based on the frequencies of user inputs, denoted as *Rank-F*. The use input with the highest frequency ranks on the top. The second baseline is a Bayesian Belief Network (BBN) [17] based ranking approach, denoted as *Rank-BBN*. Given an input parameter and a set of user contexts, we build a vector of 5 dimensions (*i.e.*, 5 ranking features in Section III-C2) for each user input in *Rank-BBN*. The vectors of user inputs are used to build a BBN. The output node is the probability of a user input being used for an input parameter. The probabilities of user inputs are used by *Rank-BBN* for ranking.

We conduct our experiment in the following steps: First, we apply *Rank-F, Rank-BBN* and our learning-to-reuse approach (denoted as *Rank-Reuse*) on *auto-data* and *manually labeled data*. We exclude the data labeled with "not sure" from our analysis. Second, we exclude the ranking

9. http://www.programmableweb.com/

feature 5 from both datasets and repeat the first step to verify the effectiveness of our approach on aiding users in *Scenario 1 and 2 of user-parameter interactions proposed in Section III-C1*. Third, both *Rank-BBN* and *Rank-Reuse* require training datasets. We follow the same data splitting strategy in [22]. We use half of a dataset for training and validation, and the remaining half for testing. Last, given a set of input parameters $P_1, P_2, \ldots, P_n$ (n is the number of input parameters), we use Equation (4) and Equation (5) to measure k-precision and k-recall for recommending a list of user inputs to P_i ($0 < i \leq n$). We use Equation (6) and Equation (7) to calculate the average precision and recall.

$$k - precision_i = \frac{|Correct\ Inputs\ in\ top\ k\ results|}{k} \quad (4)$$

$$k - recall_i = \frac{|Correct\ Inputs\ in\ top\ k\ results|}{|Correct\ Input|} \quad (5)$$

where $|\ Correct\ Input\ |$ *is a constant and = 1, because there can only be one correct input for a* P_i. $|\ Correct\ Inputs\ in\ top\ k\ results\ |$ *= (1 or 0). When k=1,* $k\text{-}precision_i$ *=* $k\text{-}recall_i$.

$$avg - k - precision = \frac{\sum_{i=1}^{n}(k - precision_i)}{n} \quad (6)$$

$$avg - k - recall = \frac{\sum_{i=1}^{n}(k - recall_i)}{n} \quad (7)$$

Table II: Performance of different approaches with different k values on *manually-labeled-data* without ranking feature 5. P and R stand for average precision and recall respectively.

Approach	WSDL		REST		Web Forms	
	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)
Rank-Reuse (k=1)	82	82	**85**	85	81	81
Rank-BBN (k=1)	58	58	55	55	62	62
Rank-F (k=1)	42	42	46	46	40	40
Rank-Reuse (k=3)	29	87	30	90	28	84
Rank-BBN (k=3)	23	69	21	63	20	60
Rank-F (k=3)	17	51	19	57	15	45
Rank-Reuse (k=5)	18	90	19	**95**	18	90
Rank-BBN (k=5)	14	70	13	65	14	70
Rank-F (k=5)	11	55	12	60	10	50

Table III: Performance of different approaches with different k values on *auto-data* without ranking feature 5. P and R stand for average precision and recall respectively.

Approach	k=1		k=3		k=5	
	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)
Rank-Reuse	**84**	84	30	90	19	**95**
Rank-BBN	65	65	23	69	14	70
Rank-F	43	43	15	45	11	55

Results. Table II and Table III show that our approach *Rank-Reuse* outperforms *Rank-BNN* and *Rank-F* in the ***Scenarios 1 and 2 of user-parameter interactions*** on both datasets. More specifically, *Rank-Reuse* achieves an average top-1 precision of 83% and top-1 recall of 83%. In most cases, the users can be saved from repetitive typing without taking actions using our Rank-Reuse. Furthermore, the high top-3 and top-5 recalls of Rank-Reuse suggest that users can identify a proper user input within top-3 or top-5 ranked values, when the users are not satisfied with the top-1 value.Moreover, the results of Table II and Table III show that automatically collected data can achieve the same level of results as the manually-labeled-data. Furthermore, our results of using *Rank-Reuse*, *Rank-BNN* and *Rank-F* on *auto-data* and *manually-labeled-data* including ranking feature 5 (*i.e.*, using one character as current typing) show that on average, the results of *Rank-Reuse* and *Rank-BNN* can be improved by 11%.

RQ 2. Which ranking features affect the ranking most?

Motivation. We investigate the effect of various features used to rank previous user inputs and discover the most influential features in ranking.

Approach. To test a feature, we first exclude the tested feature from our learning model. Second, we apply *Rank-Reuse* on *auto-data*. Third, we use Equation (6) and Equation (7) to calculate the average precision and recall for our approach. Last, we test 5 ranking features and compare the results with the ones of RQ1 to identify the effect of different features on ranking.

Furthermore, the ranking feature 3 of contexts is built on a set of contextual variables, such as location and time. To identify which contextual variables have a bigger influence on ranking, we build a new ranking feature for each type of contexts. The first feature is based on time; the second one is based on location having two contextual variables (physical locations and devices); the third one is derived from social relations having two contextual variables (who and whom). We use the same above approach of testing a feature to test the new three features.

Table IV: Effects of different types of ranking features on user input ranking. d-P and d-R stand for the decrease in average precision and recall respectively.

Test Feature	k=1		k=3		k=5	
	d-P	d-R	d-P	d-R	d-P	d-R
1 Textual	22%	22%	9%	27%	7%	35%
2 Task	7%	7%	4%	12%	4%	20%
3 Context	14%	14%	7%	21%	5%	25%
4 Frequency	5%	5%	4%	12%	3%	15%
5-Typing	11%	11%	3%	9%	2%	10%

Results. Table IV shows the effects of ranking features. The percentages in Table IV mean the decreases in the performance of Rank-Reuse without using the test feature. The bigger the percentage is, the bigger effect a ranking feature has. Without using ranking feature 1, the performance of our approach decreases most (*e.g.*, 22%-off in top-1 precision) compared with other features. Furthermore, we observe that compared with other types of contextual information, location information affects the ranking most.

V. Related Work

In this section, we summarize the related work on assigning values to services, context model of web services and learning-to-rank models.

Industrial tools such as Chrome Autofill forms [2] and Firefox Autofill Forms [3] help users fill in forms. RoboForm [23] and LastPass [24] are designed for password management and provide form auto-filling function. Academic studies (*e.g.*, [6], [14]) propose approaches for assigning values to parameters of services. AbuJarour *et al.* [6] generate annotations for web services and automatically assign values for input parameters of web services. Wang *et al.* [14] propose an automatic approach to reuse user inputs among different services by linking similar parameters and pre-filling user inputs. Our *Rank-Reuse* can be a supplement and enhancement to Wang *et al.* [14] approach.

The studies (*e.g.*, [16],[10]) propose learning-to-rank (LTR) based approaches to optimize search engines to recommend relevant documents to users. Our Rank-Reuse adopts the LTR and optimize input parameter filling process.

Mrissa *et al.* [25] propose a context model for composing services. Blake *et al.* [26] propose a context model containing user's contextual information for identifying relevant services to users. However these proposed context models are not designed for reusing user inputs.

VI. Conclusion

Unnecessary interruptions caused by repetitively typing same information to services decreases the efficiency of service composition and negatively impacts the user experience. In this paper, we propose a context-aware model for storing user inputs efficiently. We analyze four scenarios of user-parameter interactions and propose a context-aware learning-to-rank model based approach (Rank-Reuse) to pre-fill and recommend values to input parameters. Our approach learns the interactions with contextual data between user inputs and input parameters over time to reuse the user inputs for filling values to input parameters. Our empirical results show that our context-aware learning-to-reuse ranking approach outperforms two baseline approaches (*i.e.*, Bayesian Belief Network and Frequency based approaches). We observe that textual similarity based ranking feature affects the ranking most compared with other features. Among various types of contextual data, the location information (*i.e.*, physical locations and devices) matters the most to ranking.

Acknowledgment

This work is partially supported by IBM Canada Centers for Advance Studies in IBM Toronto lab.

References

[1] H. Xiao, Y. Zou, R. Tang, J. Ng and L. Nigul, *Ontology-driven Service Composition for End-Users*, Service Oriented Computing and Applications 5(3), pp.159-181,Springer,2011.

[2] Google Chrome Autofill forms,*https*://support.google.com/chrome/answer/142893. Accessed on Jan 1, 2015.

[3] Firefox Autofill Forms add-on, *https*://addons.mozilla.org /en-US/firefox/addon/autofill-forms. Accessed on Jan 1, 2015.

[4] S. Araujo, Q. Gao, E. Leonardi, J. Houben, *Carbon: domain-independent automatic web form filling*, Proc. of the Int'l Conference on Web Engineering, pp. 292-306, Vienna, 2010.

[5] S. Firmenich, V. Gaits, S. Gordillo, G. Rossi and M. Winckler, *Supporting Users Tasks with Personal Information Management and Web Forms Augmentation*, Proc. of Int'l Conference on Web Engineering, pp. 268-282, Berlin, July 23-27, 2012.

[6] M. AbuJarour and S. Oergel, *Automatic Sampling of Web Services*, Proc. of IEEE Int'l Conference on Web Services, pp. 291-298, Washington, July 4-9, 2011.

[7] T. Liu, J. Xu, T. Qin, W. Xiong, and H. Li, *Letor: Benchmarking learning to rank for information retrieval*, Proc. of the Int'l SIGIR Conference, Amsterdam, 23-23 Jul., 2007.

[8] A. Trotman, *Learning to rank*, Information Retrieval Journal, 8(3): pp.359-381, 2005.

[9] T. Y. Liu, *Learning to rank for information retrieval*, Foundations and Trends in Info. Retrieval,3(3), 225-331, 2009.

[10] T. Joachims, *Optimizing search engines using clickthrough data*. In KDD'02, 2002.

[11] L. Richardson, S. Ruby, *Restful Web Services*, O'REILLY Media, 1st edition, May 2007, ISBN-13: 978-0596529260.

[12] Sun Microsystems, *Web Application Description Language: W3C Member Submission 31*, WWW Consortium, 2012.

[13] M.F. Porter, *An algorithm for suffix stripping, Program*, 14(3) pp 130-137, 1980.

[14] S. Wang, B. Upadhyaya, Y. Zou, I. Keivanloo, J. Ng and T. Ng, *Automatic Propagation of User Inputs in Service Composition for End-users*, Proc. of the IEEE Int'l Conference on Web Services, pp. 73-80, Anchorage, June 27-July 2 2014.

[15] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, *Towards a better understanding of context and context-awareness*, Handheld and ubiquitous computing, pp.304-307, Springer, 1999.

[16] G. E. Dupret and B. Piwowarski, *A user browsing model to predict search engine click data from past observations*, Proc. of the Int'l ACM SIGIR Conference, pages 331-338, 2008.

[17] M. A. P. de Cristo, P. P. Calado, M. D. L. da Silveira, I. Silva, R. Muntz, and B. Ribeiro-Neto, *Bayesian Belief Networks for IR*, Int'l Journal of Approximate Reasoning, 34(2), 163-179.

[18] S. Lee, J. Chang, and S. g. Lee, *Survey and Trend Analysis of Context-Aware Systems*, Information-An Int'l Interdisciplinary Journal, 14(2). pp. 527-548. 2011.

[19] C. D. Manning, P. Raghavan, H. Schutze, *Scoring, term weighting, and the vector space model*, Introduction to Information Retrieval, ISBN 9780511809071, 2008.

[20] A. Singhal, *Modern Information Retrieval: A Brief Overview*, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4): 35-43, 2001.

[21] R.L. Chambers, and Skinner, *Analysis of Survey Data, Wiley*, ISBN 0-471-89987-9. 2003

[22] B. Xiang, D. Jiang, J. Pei, X. Sun, E. Chen and H. Li, *Context-aware ranking in web search*, Proc. of the Int'l SIGIR conference on Research and development in information retrieval, pp.451-458, Geneva, Jul. 19-23, 2010.

[23] RoboForm, www.roboform.com. Accessed on Feb 15, 2015.

[24] LastPass, www.lastpass.com. Accessed on Feb 15, 2015.

[25] M. Mrissa, C. Ghedira, D. Benslimane, Z. Maamar, F. Rosenberg and S. Dustdar, *A context-based mediation approach to compose semantic Web services*. ACM Trans. Internet Tech. 8(1) 2007.

[26] M. B. Blake, D. R. Kahan, M. F. Nowlan, *Context-aware agents for user-oriented web services discovery and execution*. Distributed and Parallel Databases 21(1): 39-58, 2007.