This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

Automatic Reuse of User Inputs to Services among End-users in Service Composition

Shaohua Wang, *Member, IEEE,* Ying Zou, *Member, IEEE,* Iman Keivanloo, *Member, IEEE,* Bipin Upadhyaya, *Member, IEEE,* Joanna Ng, and Tinny Ng

Abstract—End-users conduct various on-line activities. Quite often, they re-visit websites and use services to perform re-occurring activities, such as on-line shopping. The end-users are required to enter the same information into various web services to accomplish such re-occurring tasks. It can negatively impact user experience when a user needs to type the re-occurring information repetitively into such web services. In this paper, we propose an approach to prevent end-users from performing such repetitive tasks. Our approach propagates user inputs across services by linking similar input and output parameters. Our approach pre-fills values to the input parameters for an end-user using his or her previous inputs. To increase the chance of identifying a proper value for an input parameter performed by one end-user, our approach also leverages the inputs from other end-users. We identify and link similar end-users to enable the propagation of user inputs among end-users. We have designed and developed a prototype. We also conduct an empirical study to evaluate our approach using the real world services. The empirical results show that our approach using an end-user's previous inputs can reduce on average 41% of repetitive typing for the execution of composed services. Furthermore, the previous inputs from the similar end-users can improve our approach in reducing the repetitive typing for an end-user.

Index Terms—Information reuse, service composition, input parameter pre-filling, multi-user.

1 Introduction

Recently, web is playing an important role in people's life. End-users can conduct various tasks online, such as booking hotels, buying flight tickets and shopping online. Quite often, an end-user may re-visit websites or online services and compose them to perform tasks to meet his or her needs [1] [2]. For example, when an end-user wants to attend a concert, he or she buys concert tickets from *Ticket Liquidator*[1] and bus tickets from *Greyhound*[2]. These two services *Ticket Liquidator* and *Greyhound* are linked implicitly by the end-user. To invoke each service, the end-user has to provide values to the input parameters of each service. However, the current practices for assigning values to input parameters of services in service composition bring end-users the following two challenges:

- **Limitation on propagating end-user's inputs across different services.** Among the parameters of composed services, the semantically related parameters should be linked to facilitate the same information to be propagated among the relevant parameters. For example, *Ticket Liquidator* and *Greyhound* both require an end-user to input the quantity of tickets. Therefore, we should link the two input parameters requiring the same number of quantity from two services. If one of the services is filled with a numeric value (*i.e.*, the number of tickets), the value should be propagated to the other service.
- **Lack of approaches for pre-filling composed services.** End-users are often required to provide values to the input parameters of composed services. However, the information provided by the end-users can be redundant and repetitive, because the information can be previously entered into other services. For example, in the process of composing a set of RESTful services, developer account IDs issued by the service providers should be pre-filled if the end-user used the IDs before. Rukzio *et al.* [3] found that users are four times faster on a smart phone when they just correct pre-filled values compared to entering the information from scratch. Pre-filling input parameters of services can improve the efficiency of service invocation.

To improve the user experience in service composition, there is an urgent need to call for an approach that prevents the end-users from interruptions caused by unnecessarily data entries to services. Recently, several approaches have been developed to fill in web forms. Google Chrome Autofill [4] fill in web forms using end-user's previous inputs. Araujo *et al.* [5] propose a concept-based approach for automatic web form filling. These approaches treat input parameters of services individually, and do not identify and link the relevant parameters for assigning values to input parameters of services. AbuJarour *et al.* [6] propose an automatic approach to assign values to input parameters of web services. Thomas *et al.* [7] and Gerede *et al.* [8] identify input-output data flows of services. However, all of the above approaches heavily rely on the names of parameters for identifying similar parameters. When the

- *Shaohua Wang is with the School of Computing, Queen's University, Kingston, Ontario, Canada. E-mail: shaohua@cs.queensu.ca*
- *Ying Zou, Iman Keivanloo and Bipin Upadhyaya are with the Electrical and Computer Engineering, Queen's University, Kingston, Ontario, Canada. E-mail: {ying.zou, iman.keivanloo, bipin}@queensu.ca*
- *Joanna Ng and Tinny Ng are with the IBM CAS Research, Markham, Ontario, Canada. E-mail: {jwng, tng}@ca.ibm.com*

1. *http://www.ticketliquidator.com*
2. *http://www.greyhound.ca/*

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

names of parameters are not suitable for calculating the similarity between input parameters due to bad naming and incorrect spelling, these approaches do not perform well. Moreover, all of the above approaches are not designed for filling values to input parameters in the context of service composition.

In this paper, we propose an approach that captures user's previous inputs, propagates the inputs across different services among multiple end-users, and pre-fills input parameters during the service composition. We propose a context-aware Meta-data Model for capturing user's inputs. The Meta-data Model is designed to store the task information of a user input (*i.e.*, a value to an input parameter). We propose an Input Parameter Context Model to identify similar parameters of services in order to reuse user's inputs across services. Our approach uses the context-aware Meta-data Model and Input Parameter Context Model for context-aware matching between the previous user inputs and the input parameters during the service compositions.

In some cases, one end-user's previous inputs may not be sufficient to identify a proper value for pre-filling an input parameter due to the lack of history. To increase the chance of discovering a suitable value for an end-user, instead of only using his or her previous inputs, we collect and leverage user inputs from other end-users who share the similar activities as the end-user does. Two end-users are similar if the two end-users have performed similar activities before. We calculate the similarity between two end-users using their historical activities (*e.g.*, services performed). Our approach identifies a set of similar end-users to an end-user, and leverages the user inputs from the identified similar end-users. Our approach allows us to maximize the reuse of user's inputs and take our best effort to prevent end-users from being interrupted by redundant data entries during the service execution.

In our previous work [9] published in the proceedings of 21^{st} International Conference on Web Services, we propose an approach that captures user's previous inputs, propagates the inputs across different services, and pre-fills input parameters during the service composition. The approach is designed to reuse one end-user's previous inputs. In this paper, we extend the earlier work [9] in the following aspects:

1) We redefine the pre-filling model by adding the dimension of multi-users into the model. The proposed approach for input parameter filling in the earlier work is only applied to one end-user. The extended approach can leverage and consume the user inputs from multiple end-users.
2) We propose an approach for identifying similar end-users to an end-user. It is critical to identify similar end-users in order to propagate the user inputs among end-users.
3) We conduct an empirical study of the effectiveness of our approach for identifying similar end-users. The empirical results show that in 83% of the cases, our approach can identify the most similar end-user to an end-user.
4) We propose an approach to reduce the number of input parameters that an end-user is required to provide values to by taking into consideration the user inputs from his or her similar end-users.
5) We conduct an empirical study of the effectiveness of our overall approach for reducing the input parameters by leveraging user inputs from different end-users. The empirical results confirm that including user inputs of similar end-users of an end-user can help reduce the parameter filling for the end-user.

The rest of this paper is organized as follows. Section 2 presents the structures and background of web services. Section 3 introduces the proposed approach. Section 4 introduces the case study on our approach. Section 5 discusses the threats to validity. Section 6 summarizes the related literature. Finally, Section 7 concludes the paper and outlines some avenues for future work.

2 Web Services

Our paper is primarily focused on reusing end-user's inputs among web services during the service composition. A web service is a software module designed to help interoperation between machines over the web. There are two types of specifications for exchanging information among web services, namely Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) [10]. Web Services Description language (WSDL) [11] is an XML-based interface description language for describing the functionality of a web service (*i.e.*, SOAP-based services), and often used in combination with SOAP. RESTful services [12] simplify the development, deployment and invocation of web services. Compared with SOAP-based services, RESTful services are light weight. RESTful services use standard HTTP and permit different data formats. Web Application Description Language (WADL) [13] is an XML-based description of HTTP-based web applications (*i.e.*, typically REST web services). WADL is the REST equivalent of WSDL services.

The end-users can also conduct various tasks through web applications (*e.g.*, web forms). In this paper, we consider a web application as a type of on-line services. We treat SOAP-based and RESTful services as the services with descriptions and web forms of web applications as the services without formal descriptions.

In a service invocation chain, there are three types of linkages [14]: 1) User-to-Service occurs when a user invokes a service; 2) Service-to-Service occurs when a service invokes another service; and 3) Service-to-User occurs when the service needs human inputs or confirmation. In this study, our approach facilitates these three linkages.

3 Overview of Our Approach

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

1) *Collecting and storing user inputs*. The user's previous inputs can be an excellent source to feed input parameters. A user input is a piece of information entered by an end-user into services. To exploit such information, the first step is to capture such data properly for further usage.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

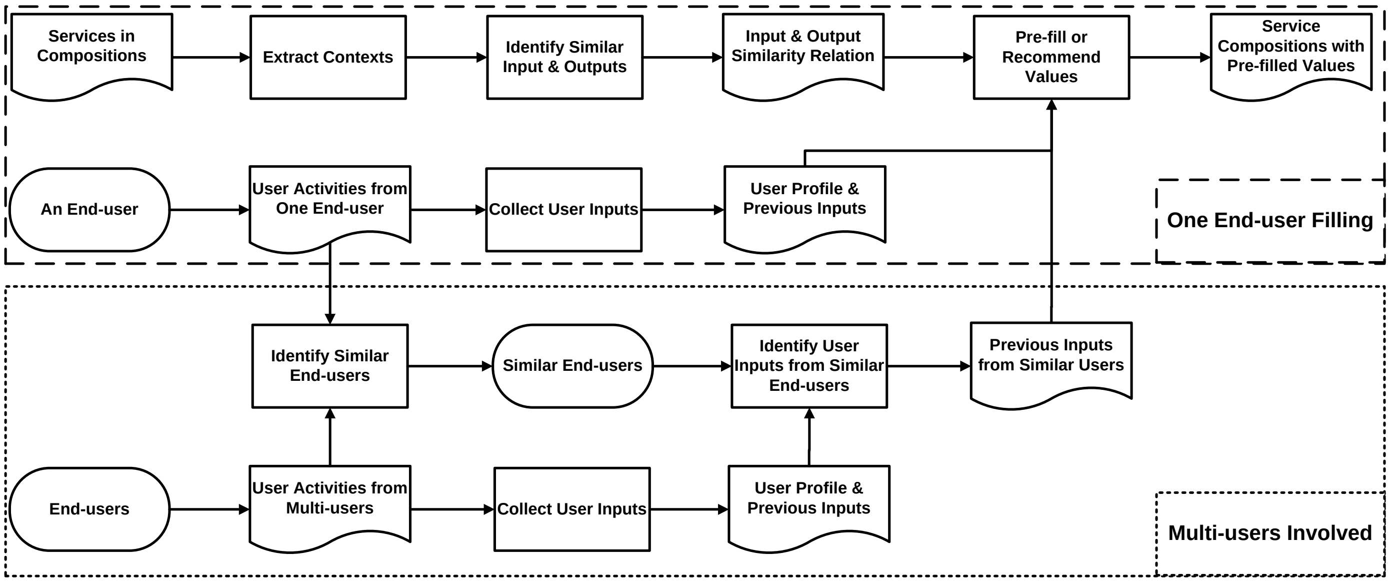

Services in Compositions
Extract Contexts
Identify Similar Input & Outputs
Input & Output Similarity Relation
Pre-fill or Recommend Values
Service Compositions with Pre-filled Values
An End-user
User Activities from One End-user
Collect User Inputs
User Profile & Previous Inputs
One End-user Filling
Identify Similar End-users
Similar End-users
Identify User Inputs from Similar End-users
Previous Inputs from Similar Users
End-users
User Activities from Multi-users
Collect User Inputs
User Profile & Previous Inputs
Multi-users Involved

Fig. 1: Overall steps for propagating the inputs of end-users across different services during service composition.

2) *Collecting contexts of parameters and building input-output dependency*. We extract the contexts of parameters to identify similar parameters for building input-output parameter dependency. The three contexts of an input parameter are *Textual* describing the textual information of the parameter, *Task* defining the task information of the parameter, and *Neighbors* storing the information of a group of other parameters locating in the same operation as the parameter. The dependency contains the semantic relations among parameters. The dependency is used to propagate user's inputs across different services.
3) *Identifying similar end-users*. Two end-users are similar if these two end-users perform similar or identical tasks, such as services composed. Similar end-users may fill in the same value to an input parameter. We link the end-users conducting similar activities, such as the services composed and input parameters provided with values in the composed services.
4) *Collecting user inputs from similar end-users*. For an end-user required to enter a value to an input parameter of a task, we identify the similar or identical tasks from the similar end-users identified in the step of *Identifying similar end-users*. Then, we collect the user inputs associated with the identified tasks for pre-filling the input parameter.
5) *Pre-filling values for operations of services*. This step selects a proper value for an input parameter from all of the possible data sources: an end-user's personal information and his or her similar end-users' previous inputs. The end-user's personal information includes his or her profile information and his or her previous inputs.

The Steps 1), 2) and 3) are the main steps for pre-filling input parameters using one end-user's previous inputs and profile information. With the addition of the steps of *Identifying similar end-users* and *Collecting user inputs from similar end-users*, our approach can use not only one end-user's previous inputs, but also other end-user's previous inputs for pre-filling input parameters.

3.1 Five Scenarios of Assigning Values to Input Parameters

We identify five scenarios of supplying a value to an input parameter of a service participating in a service composition performed by an end-user. The five scenarios are listed as follows:

1) *Scenario 1:* Sharing values among the similar input parameters of operations. The operations can be from a single service or different services. If two input parameters from different operations are semantically related, the value provided to one of operations can be propagated to fill in the other one ether in the same service or a different service.
2) *Scenario 2:* Reusing values of output parameters to fill in an input parameter. The output parameters are generated from an operation, and the targeted input parameter is requested from another operation. The operations can be either from a single service or different services. The output of an output parameter can be used to fill in an input parameter. For example, an operation of a service generates a city name using zip code, and an input parameter of a weather forecast service requires a city name to provide weather forecast.
3) *Scenario 3:* Using other data sources of an end-user to assign values to input parameters. The personal profile data and the previous user inputs from the end-user are used as the external data sources to fill in an input parameter of a service performed by the same end-user. The end-users could execute the similar services consuming the previous inputs or information in personal profile. For example, an end-user may conduct a business trip multiple times

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

within a year with same inputs except for departure and return dates.

4) *Scenario 4:* Leveraging the user inputs from other end-users similar to an end-user. An end-user' previous inputs may be not sufficient to pre-fill any encountered input parameters. However, using the user inputs from the similar end-users of the end-user could increase the chance of pre-filling the right values to the encountered input parameters. The end-users who perform similar activities could share and reuse their inputs among them.
5) *Scenario 5:* Requiring data entry by end-users. If the aforementioned sources are not available, the end-users have to enter the values manually.

In this study, we collect user inputs from end-users when they use web interfaces and our approach uses the collected information in the above five scenarios.

3.2 Collecting and Storing User Inputs from End-users

A user input is a piece of information entered by end-users into services. It is usually stored as a key-value pair by existing approaches such as [4] [5] [15] where the key is the type of the information and the value is the information itself, for example "address" is the key and "Kingston, Canada" is the value. The key-value model lacks of information for conducting context-aware matching between the values and input parameters. For example, there are two values for a key "contact phone number", such as (647)222-3333 and (123)456-7890; the end-users use (647)222-3333 for flight booking and (123)456-7890 for hotel reservation. We need to modify and enrich the key-value pair model by attaching contextual information to the piece of information and propose a context-aware Meta-Data Model for storing user inputs.

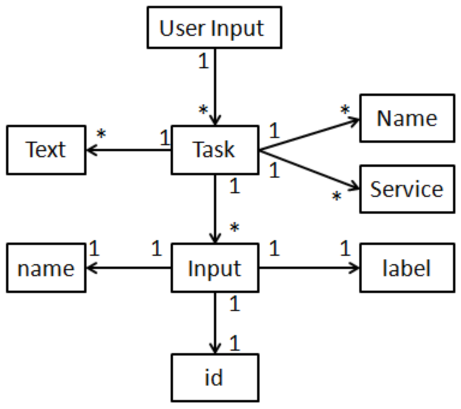


Fig. 2: Description of Context-aware Meta-Data Model

3.2.1 *Context-aware Meta-Data Model of User Inputs*

Figure 2 shows the description of our proposed context-aware meta-data model. Figure 3 illustrates an example of storing an email address "shaohua@cs.queensu.ca" entered into the login service of Ebay[3]. A user input has two properties: Type and Task. The Type property records the data type of the information. A task can be an operation in WSDL services, or a resource of a RESTful service and the associated action (*e.g.*, get or delete) or a web form of a web application. As shown in Figure 3, the data type of "shaohua@cs.queensu.ca" is email and its task is "Sign in". A user input can be associated with multiple tasks. For example, an address 25 Union St. Kingston, Canada can be used as a shipping address, and also a home address for different tasks. The Task property describes the applicable tasks for a user input (*i.e.*, a piece of information) and has four sub-properties listed as follows:

- ***Name Property*** states the name of a task, such as the name of an operation defined in WSDL, the resource name and its associated actions of a RESTful service, and a label (*e.g.*, from a web form or HTML tag). As shown in Figure 3, the Name Property of the task of "shaohua@cs.queensu.ca" is"Sign in".
- ***Text Property*** defines the description of a task. We retrieve a description for a task from the description of an operation in WSDL, the descriptive text of a resource in a RESTful service, or textual information of a web form. If the description is not available, we assign NULL to this property. In Figure 3, the Text Property of the task "Sign in" is web form name "SignInform".
- ***Service Property*** records the name of a service in which a task is performed. We use the name of the WSDL, RESTful service, or the URL of a web form as a service property. As shown in Figure 3, the service property is "URL of the login Web form".
- ***Input Property*** stores the coding information of the input parameters. The coding information includes three properties: *label*, *name*, *id*. In the context of Web applications, *label*, *name*, *id* are the attributes of an HTML DOM element which defines the input field consuming the information. As shown in Figure 3, the values of input properties: *label*, *id* and *name* are "Email or user ID", "userid" and "userid". If the information is entered into a WSDL or RESTful service, we store the name of the input parameter in the property of *label*.

3.2.2 *Collecting User Inputs*

The end-user's inputs can be collected through end-user's activities, such as on-line activities and web service testing. For example, the end-user's on-line activities can be shopping on-line and filling in student registration web forms; the web service testing can be testing a RESTful service using its URL through a web browser, or testing a SOAP-based WSDL service using SOAPUI framework[4].

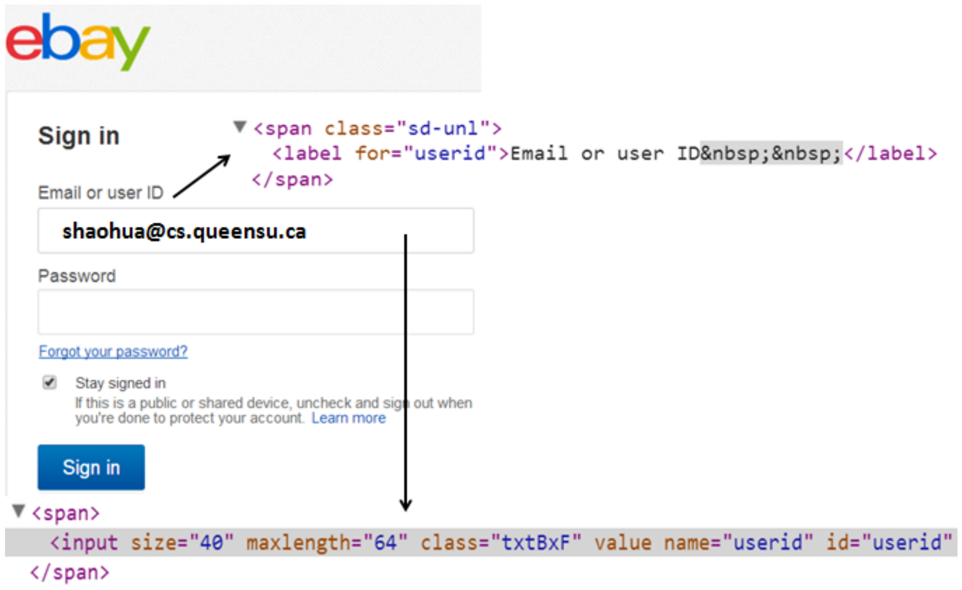
We modify and extend an open source tool called Sahi[5] to monitor end-user's web activities. Sahi injects Javascripts into web pages and monitors the user's actions (*e.g.*, click, submit), records the user's inputs. We extend the Sahi to include the detail information of a user interface component where the end-user enters a value. Then we mine the information of properties as specified in Section 3.2 from Sahi log

3. *www.ebay.com*

4. http://www.soapui.org/
5. http://sahi.co.in/

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278


Sign in
Email or user ID
shaohua@cs.queensu.ca
Password
Forgot your password?
Stay signed in
If this is a public or shared device, uncheck and sign out when you're done to protect your account. Learn more
Sign in

```
▼ <span class="sd-unl">
    <label for="userid">Email or user ID&nbsp;&nbsp;</label>
  </span>
```

```
▼ <span>
    <input size="40" maxlength="64" class="txtBxF" value name="userid" id="userid"
  </span>
```

Fig. 3: A mapping between UI and its HTML coding.

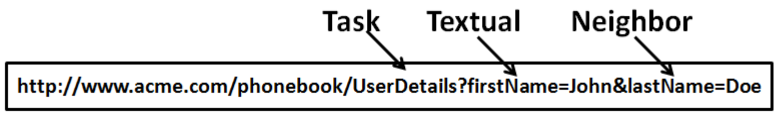

Task
Textual
Neighbor
http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe

Fig. 4: An example annotated screenshot of showing the four contexts of a parameter from a RESTful service.

and store the user inputs with their property information in the Meta-Data Model. Our tool is a standalone application running on different operating systems including Windows 7 or 8 and Linux OS. The end-users can run our tool and surf the web through web browsers including Google Chrome[6] and Mozilla Firefox[7].

3.3 Collecting Contexts of Parameters and Building Input-Output Dependency

To propagate an end-user's input across services and reduce the number of input parameters requiring user inputs, it is essential to link similar inputs and output parameters across services. The two similar input parameters from different operations (*i.e.*, **Scenario 1** as described in Section 3.1) could require the same values. The value of an output parameter of an operation can be consumed by an input parameter of another operation (*i.e.*, **Scenario 2** as introduced in Section 3.1). We exclude fault messages, such as "error code", for services in our analysis as they seldom contribute to the data flow in the subsequent services.

3.3.1 Input Parameter Context Model

We propose an Input Parameter Context Model to identify the linkages among parameters. We extract three contexts for a parameter denoted as P (*i.e.*, an input or output parameter) and calculate the similarity between parameters using their contexts. Figure 4 illustrates an annotated screenshot showing the contexts of a parameter from a RESTful service. Figure 5 shows the description of our proposed Input Parameter Context Model. The three contexts of a parameter are listed as follows:

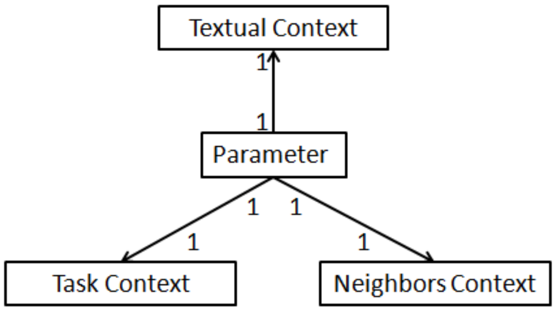


Fig. 5: Description of Input Parameter Context Model

- **Textual Context** (denoted as P^{Text}) defines the textual information of a parameter. In most cases, the value is the name of a parameter from an operation defined in WSDL and RESTful services. For example, the textual information of the input parameter "firstName" in Figure 4 is "firstName". RESTful services defined in WADL[8] usually provide some descriptive text for an input parameter. For an input parameter from Web component service, we extract text from *label*, *id* and *name* from the HTML DOM [16].
- **Task Context** (denoted as P^{Task}) stores the task information. A task is an operation in a WSDL service, a resource and its associated actions (*e.g.*, get or

6. http://www.google.ca/chrome/
7. https://www.mozilla.org/en-US/
8. http://www.w3.org/Submission/wadl/

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

delete), or a web form of a web application. We store the descriptive information along with a name for the task. For example, the task name of the input parameter "firstName" in Figure 4 is "UserDetails". The descriptive information of the task "UserDetails" can be obtained from the web page where the resource of a RESTful service is described.

- **Neighbors Context** (denoted as $P^{Neighbor}$): stores a group of other parameters required by the same operation. For example, the task "UserDetails" in Figure 4 has two input parameters: "firstName" and "lastName". Then, the two parameters are neighbors of each other.

After extracting contexts for a parameter, we store them in the following format $P =< P^{Text}, P^{Task}, P^{Neighbor} >$.

3.3.2 Building Input-Output Similarity Relation

To identify meaningful words from the contexts, we conduct word normalization. We decompose any possible compound words (*e.g.*, FindCity). The naming of operations or parameters follows 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). To remove non-English words, we use WordNet [17] which is a lexical database organizing the English words into a set of synonyms. Furthermore, we remove the possible stop words using a pre-defined list of stop words, such as "the". Finally, we use porter stemmer [18] to reduce derived words to their stem, base, or root form (*e.g.*, "reserves" and "reserved" are mapped to the stem form "reserve").

After the word normalization, we calculate the context similarity. A context, such as task information P^{Task} of a parameter, could contain more than one word (*e.g.*, "book car"). We formulate a context into a phrase, Ph, which is a collection of words, $Ph = \{W_1, W_2, \ldots, W_n\}$, where n is the number of words in Ph. We use WordNet to calculate the semantic similarity between two phrases Ph_i and Ph_j in the following cases:

- Case 1. We identify the common words (*i.e.*, two words are identical or synonymous) of two phrases. If Ph_i and Ph_j contain the same number of words and all of the words are same, then Ph_i=Ph_j.
- Case 2. If the number of words contained in Ph_i is greater than the number of words contained in Ph_j, and all of the words in Ph_j are contained in Ph_i, we use Equation (1) to calculate the similarity between two phrases.

$$sim(Ph_i, Ph_j) = \frac{|Ph_i \cap Ph_j|}{|Ph_j|} \quad (1)$$

  where $sim(Ph_i, Ph_j)$ denotes the semantic similarity value between Ph_i and Ph_j, Ph_j is a subset of Ph_i.
- Case 3. If there is no *containment* relation between Ph_i and Ph_j, we use the following steps to calculate the semantic similarity between two phrases.
  - Step 1: We calculate the number of pairs among common words of two phrases Ph_i and Ph_j.
  - Step 2: We calculate the semantic similarity between every pair of words W_a ($W_a \subset Ph_i$) and W_b ($W_b \subset Ph_j$) from two phrases after excluding the common words from calculation. Then, we sum up all of the similarity values of pairs of words, which is denoted as $Sum_{sim} = \sum_{W_a \subset Ph_i} \sum_{W_b \subset Ph_j}$ sim(W_a, W_b), where *sim(W_a, W_b) denotes the similarity value between two words, W_a and W_b; W_a and W_b are not identical*.
  - Step 3: We use Equation (2) to calculate the similarity between two phrases. The numerator of the equation is to calculate the similarity between each pair of words of two phrases; the denominator of the equation is the total number of times of calculating the similarity between words.

$$sim(Ph_i, Ph_j) = \frac{|Ph_i \bigcap Ph_j| + Sum_{sim}}{|Ph_i \bigcap Ph_j| + |pairs\ of\ words\ without\ common\ words|} \quad (2)$$

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

Given two parameters, P_i and P_j, for similarity calculation sim(P_i, P_j), we first calculate the similarity between each of their contexts separately using Equation (2), then we integrate the similarity of each context of parameters using Equation (3).

$$sim(P_i, P_j) = M_a * sim(P_i^{text}, P_j^{text}) + M_b * sim(P_i^{task}, P_j^{task}) + M_c * sim(P_i^{neighbor}, P_j^{neighbor}) \quad (3)$$

where $sim(P_i, P_j)$ defines the similarity between two parameters.

$$sim(P_i, P_j) = \begin{cases} 1 & \text{if they are identical} \\ 0 & \text{if they are completely different} \end{cases}$$

$sim(P_i^{text}, P_j^{text})$ defines the semantic similarity between the context of Textual Information from two parameters, $sim(P_i^{task}, P_j^{task})$ is the semantical similarity between the context of Task Information of two parameters, $sim(P_i^{neighbor}, P_j^{neighbor})$ evaluates the semantic similarity between the context of Neighbors of two parameters. M_a, M_b and M_c are the weights of a context. We assign equal weights to each context, i.e., 0.33. The weights of a context are derived empirically.

Given a set of services $S_1, S_2, \ldots, S_n$ (n is the number of services) in an on-the-fly service composition, we extract the input and output parameters of services. Then, we calculate the similarities between each input parameter and all of the other input and output parameters using contexts of the parameters. Therefore, each input parameter has a vector of similarity values. We remove the similarity values which are less than a threshold value which is derived empirically.

Given two services, S_i and S_j ($1 \leq i < j \leq n$) of a service composition, S_j is composed after S_i. To fill in an input parameter, P of an operation in the service S_i, the parameters of operations from the service S_j cannot

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

be used as data sources for S_i, because the parameters in the service S_j are from subsequent services. We remove such parameters from our analysis when we identify the similar parameters for the input parameter P. We rank the similarity values from the highest to the lowest within the vector for each input parameter. We build input-output similarity relation based on the vectors of similarity values among parameters.

Using the input-output similarity relation, we can reduce the number of input parameters requiring user inputs by identifying the similarity relations between parameters in **Scenario 1** and **Scenario 2**.

3.4 Identifying Similar End-users

An end-user's previous inputs and personal profile may not be enough to pre-fill proper values to input parameters. To increase the odds of pre-filling a proper value to an input parameter for one end-user, we leverage other end-users' historical information (*i.e.*, previous inputs) for pre-filling input parameters. When a group of the similar end-users perform the similar tasks, they could reuse the same inputs. Linking similar end-users is a critical step to reuse the user inputs for end-users. In this section, we introduce our approach of identifying similar end-users for an end-user.

An end-user U performs a set of tasks $\{Task_1^U, Task_2^U, \ldots Task_n^U\}$, where n is the number of tasks performed by the end-user. Each task, $Task_i^U$ ($1 \leq i \leq n$), involves a set of services and input parameters entered with values. Given two end-users U_i and U_j, we calculate the similarity between them in the following steps:

- Step 1: We identify common services and input parameters entered with a value of two end-users, U_i and U_j. We use Equation (4) to calculate the number of common services and input parameters between two end-users.
$$Common(U_i, U_j) = |Common\ Services| + |Common\ Parameters| \quad (4)$$
where $Common(U_i, U_j)$ defines the number of common activities between two end-users, U_i and U_j.
- Step 2: We identify the end-user having more services and parameters, and then count the number of services and parameters. We use Equation 5 to calculate the number of services and parameters from the end-user having more activities.
$$Longer = |Services| + |Parameters| \quad (5)$$
*where **Longer** defines the total number of services and parameters from the end-user having more activities.*
- Step 3: We use Equation (6) to calculate the similarity value between U_i and U_j.
$$sim(U_i, U_j) = \frac{Common}{Longer} \quad (6)$$

where $sim(U_i, U_j)$ defines the similarity between two end-users and ranges from 0 to 1.

$$sim(U_i, U_j) = \begin{cases} 1 & \text{if they are identical} \\ 0 & \text{if they are completely different} \end{cases}$$

To identify similar end-users to an end-user U, we check all of the other end-users and calculate the similarity value between any other end-user and the end-user U. Then we select the other end-user with the highest similarity value as the similar peer to the end-user U.

3.5 Collecting User Inputs from Similar End-users

To identify the most likely useful user inputs from the similar end-users to pre-fill an input parameter P for an end-user, we identify the same or similar input parameters performed by both the end-user and his or her similar peers.

Given an end-user U and one of his or her similar peers U_p, we obtain the possible user inputs from a similar peer U_p in the following steps:

- Step 1: We traverse all of the tasks $\{Task_1^{U_p}, Task_2^{U_p}, \ldots Task_n^{U_p}\}$ of the similar peer U_p to verify whether the tasks are similar or identical to the task $Task_{current}^U$ performed by an end-user U. We calculate the similarity between each task, $Task_i^p$ ($1 \leq i \leq n$), and the current task $Task_{current}^U$ using the approach of calculating the similarity value between two phrases described in Section 3.3.
- Step 2: We select the task $Task_i^{U_p}$ of the similar peer U_p with the highest similarity value as the matching task to the current task $Task_{current}^U$ of the end-user U.
- Step 3: We traverse all of the input parameters of the task, $Task_i^{U_p}$ selected in the previous step. We calculate the similarity value between each input parameter of the selected task $Task_i^{U_p}$ and the input parameter P performed by the end-user U using Equation (3).
- Step 4: We select the input parameter in **Step 3** with the highest similarity value, and then use the user inputs associated with the selected input parameter as the collected user inputs from U_p. We exclude user's personal information from the collected user inputs, because the personal information, such as end-user's name and shoe size, can not be reused in a different end-user context.

3.6 Pre-filling Values for Input Parameters of Services

In this section, we describe how our approach handles the **Scenario 3** and **Scenario 4** as discussed in Section 3.1 to provide values to input parameters in order to maximize the reuse of user inputs across services.

To fill in an input parameter P, we identify a proper value from the previous user inputs stored in the context-aware Meta-Data Model proposed in Section 3.2.1 and the collected user inputs from similar end-users presented in Section 3.5 if the end-user has no matching parameters from his or her previously used services. We traverse all of the available user inputs and calculate the similarity value between each available user input and the input parameter P. We select the user input with the highest similarity value as the input for the requested input parameter P.

Given a user input I and an input parameter P, we calculate the semantic similarity between them. We conduct word normalization on all of the properties of the user

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

input I using the approach as discussed in Section 3.3. A user input can be associated with multiple tasks in different contexts. Therefore, we traverse all the possible ***Tasks***, $\{Task_1^I, Task_2^I, \dots Task_n^I\}$, associated with the input I to verify whether the input I can be used to the input parameter P (where n is the number of tasks that the user input is used for). We calculate the similarity between a task associated with the user input I, denoted as $Task_i^I$ ($1 \leq i \leq n$), and the input parameter P in the following steps:

1) We combine the words used in the ***Label***, ***ID*** and ***Name*** in the ***Input Property*** of $Task_i^I$ (a task associated with the user input I) as the input information of $Task_i^I$, denoted as $IP(Task_i^I)$. We use Equation (1) proposed in Section 3.3 to calculate the similarity value between the $IP(Task_i^I)$ and the ***Text Context*** of the input parameter P, denoted as $Text_p$. The similarity calculation is denoted as sim($IP(Task_i^I)$, $Text_p$).
2) We merge the words from ***Text and Name properties*** of $Task_i^I$ as Textual information, which is denoted as $Text(Task_i^I)$. We use Equation (1) proposed in Section 3.3 to calculate the similarity value between $Text(Task_i^I)$ and the ***Task context*** of the input parameter P, denoted as $Task_p$. The similarity calculation is denoted as sim($Text(Task_i^I)$, $Task_p$).
3) We use Equation (7) to calculate the similarity between a ***Task Property*** of input I and an input parameter P.

$$sim(Task_i^I, P) = M * sim(IP(Task_i^I), Text_p) + N * sim(Text(Task_i^I), Task_p) \tag{7}$$

M is the weight for the Input Similarity and N is the weight for the Textual Similarity between the user input and the input parameter. We assign 0.5 to M and 0.5 to N. The weights are derived empirically. $sim(Task_i^I, P)$ *defines the similarity between the* ***Task Property*** *of the user input and the input parameter.*

$$sim(Task_i^I, P) = \begin{cases} 1 & \text{if perfectly matched} \\ 0 & \text{if completely not matched} \end{cases}$$

We select a $Task_j^I$ ($1 \leq j \leq n$) with the highest similarity value with the input parameter as the match for the input parameter.

4 Case Study

We introduce our case study setup and the research questions. For each question, we present the motivation behind the question, the analysis approach and our findings.

4.1 Case Study Setup

We conduct our study on two types of services: SOAP-based services & RESTful with service description, and web applications without service description. In our dataset, the SOAP-based services are in WSDL; the RESTful services are described in web pages.

TABLE 1: Descriptive Statistics About Our Dataset of Public Web Services.

Domain	# of WSDL and REST	# of Web forms
Travel	235	30
E-commerce	192	15
Finance	164	10
Entertainment	109	10

Collecting public web services: In total, we download and collect 640 public available WSDL files. We use programmableWeb[9] to collect 60 URLs of RESTful services and download the web pages containing the description of APIs. We manually collect the contextual information related to RESTful services, such as the description of resources. The 700 services with service descriptions fall into 4 domains: Travel (*e.g.*, book flights), E-commerce (*e.g.*, buying shoes), Finance (*e.g.*, check a stock price) and Entertainment (*e.g.*, search TV shows). We use Google to search for websites for each domain to download web forms. We choose the websites listed on the top of the Google result sets. In total we download 50 web forms. Table 1 provides a summary of the dataset used in our case study.

Collecting user inputs and user activities: To evaluate the effectiveness of our pre-filling approach, we require a set of recorded data for input fields. We collect user inputs through our input collector proposed in Section 3.2. We recruit five graduate students who typically spend 8-10 hours per day on-line in addition to the first author of this paper for user inputs collection. The six subjects (*i.e.*, five recruited graduate students and the first author of this paper) used the tool proposed in Section 3.2 to track their inputs and activities for three days (*e.g.*, which web forms and input parameters used) on web forms requiring their information, such as name, email, address, and phone number. Furthermore, we use different values for diverse tasks. For example, we use different email addresses for various tasks such as login, contact information.

To test the effectiveness of our approach for linking similar end-users, we manually identify the most similar subject to a subject among the other five subjects and label the identified subject. Each subject has a labeled most similar subject. We use the 6 labeled subjects as the gold standard to verify the results of our approach.

4.2 Research Questions

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

RQ 1. Can the proposed Input Parameter Context Model help identify similar parameters?

Motivation. Linking similar parameters is critical to propagate end-user inputs across services. In this question, we measure the effectiveness of identifying similar parameters using our proposed Input Parameter Context Model (IPCM).

Approach. We build a baseline approach that uses the names of parameters to identify similar parameters. We compare the baseline approach with our approach using

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

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

Input Parameter Context Model to identify similar parameters.

We collect all of the input and output parameters from 700 WSDL and RESTful services, and 60 web forms separately. Then, we calculate the similarity value between each input parameter and the other input or output parameters using our approach and the baseline approach. Our approach and the baseline approach identify the most similar parameter (*i.e.*, input or output parameter) for each input parameter. Therefore, the pairs of parameters are generated from both our approach and the baseline approach. Each pair has an input parameter and its most similar parameter (*i.e.*, the parameter has the highest similarity value with the input parameter).

TABLE 2: Results of the Accuracy of Our Approach and the Baseline Approach for Identifying Similar Parameters.

Approach	WSDL and REST	Web Forms
Our approach	84%	89%
Baseline	68%	61%

We randomly sample the pairs of parameters collected from our approach and the baseline approach with confidence level 95% [19] for manual verification of the results. We randomly sampled 376 pairs generated from WSDL and REST services and 230 pairs generated from web forms, and manually verify the correctness of the pairs of parameters for each approach. We use Equation (8) to measure the accuracy of our approach and the baseline approach.

$$Accuracy = \frac{Number\ of\ Correct\ Pairs}{Number\ of\ Parameter\ Pairs} \tag{8}$$

Results. Table 2 shows the results of our evaluation. Our approach achieves an effectiveness of 84% and 89% on WSDL and RESTful services and web applications respectively. Our approach outperforms the baseline approach. The baseline approach heavily relies on the names of input parameters. If the names are not available or the words of names are not meaningful, the baseline approach fails in linking the services. Even though the name of parameters cannot help calculate semantic similarity, our approach can still identify the similar parameters because our approach uses three contexts of input parameters.

RQ 2. Is our proposed pre-filling approach effective to fill in input parameters?

Motivation. It is essential to save end-users from repetitive data entries by pre-filling values to input parameters of operations of services. We are interested in evaluating our pre-filling approach using the task information of user inputs and the contexts of parameters. The task information of user inputs is stored in the proposed context-aware Meta-data Model proposed in Section 3.2.1. The contexts of parameters are described in the Input Parameter Context Model proposed in Section 3.3.1.

Approach. Pre-filling requires the similarity calculation between a user input and an input parameter to determine if there exists any opportunity for pre-filling. In this paper, the user inputs are described using the context-aware Meta-data Model (MDM) and the contexts of a parameter are in the Input Parameter Context Model (IPCM). Our pre-filling approach uses MDM and IPCM to calculate the similarity between a user input and an input parameter. To compare with our approach, we build a baseline approach based on the key-value pair model as mentioned in Section 3.2 to describe an end-user's input and the name of an input parameter.

To validate the results manually, we randomly sample the input parameters with the confidence level of 95% [19]. We randomly select 376 input parameters from WSDL and RESTful services and 230 input parameters from web forms. Both our approach and the baseline approach use the collected end-users' inputs using our input collector to fill in the selected input parameters. To measure the effectiveness of our approach, we calculate precision and recall using Equation (9) and Equation (10).

$$Precision = \frac{|Correct\ Filled\ Input\ Parameters|}{|Filled\ Input\ Parameters|} \tag{9}$$

$$Recall = \frac{|Correct\ Filled\ Input\ Parameter|}{|Input\ Parameters\ Need\ To\ Be\ Filled|} \tag{10}$$

Results. Table 3 shows the results of our evaluation. Our approach outperforms the baseline approach. The recalls of the two approaches on WSDL and REST are relatively low, because the input parameters are randomly selected from 700 services and most of them do not require the collected end-users' inputs, such as email. However, web forms are usually designed to require the basic personal information. The baseline approach cannot distinguish the information related to different tasks. For example, a key *"contact phone number"* in the key-pair model can have multiple values, one value is used as a travel contact number and the other value is used as the contact number of receiving shipments from E-commerce websites. The baseline approach can only achieve a precision of 55% on average. However, our pre-filling approach achieves a precision of 78% on average.

RQ3. Can our approach using previous user inputs of an end-user reduce the number of inputs required from the end-user?

Motivation. RQ1 and RQ2 measure the effectiveness of two different sub-steps of our overall approach. In this question, we evaluate the effectiveness of our overall approach for reducing the number of inputs which an end-user is required to enter in a single user environment. In this research question, we do not leverage the user inputs from similar end-users who performed the same task during the service composition.

Approach. To answer the research question, we create a set of composed services. More specifically, we combine web forms within a web application to form service compositions. For WSDL and RESTful services, we randomly select 30 services from each domain, and use the data flow dependency to link the selected services. Then, we remove the compositions which only have one service in the service compositions. To test the effectiveness of our approach in reducing repetitive typing required for an end-user during service composition, we randomly select 5 service compositions from the generated compositions from each domain for each type of services (WSDL and REST, and web forms). To avoid errors in the compositions generated by an automatic

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

TABLE 3: Results of the Evaluation of Our Approach and the Baseline Approach of Pre-filling Values to Input Parameters.

Approach	WSDL and REST		Web Forms	
	Precision(%)	Recall(%)	Precision(%)	Recall(%)
Our Approach	75	18	81	42
Baseline	42	10	68	34

approach, we manually correct the errors to remove the possible bias in our observations.

To test the effectiveness of our approach for reducing the number of input parameters in a single user environment, we exclude the steps of *Identifying Similar End-users* and *Collecting User Inputs from Similar End-users* which are related to a multi-user environment. We apply our approach on the randomly selected compositions of services. Our overall approach analyzes the parameters and generates a list of input parameters which need end-user's inputs. Our approach uses the collected user inputs of each subject (*i.e.*, discussed in Section 4.1) to fill in the input parameters. We use Equation (11) to calculate the effectiveness of our approach for each subject. The nominator of Equation (11) calculates the difference of the numbers of input parameters requiring user inputs between without using our approach and using our approach on the selected compositions; the denominator of Equation (11) is the number of input parameters requiring user inputs without using our approach. Last, we use Equation (12) to measure the average effectiveness of our approach for six subjects.

$$Effectiveness_i = \frac{Before\ -\ After}{Before} \tag{11}$$

where **Before** *stands for the number of input parameters requiring end-user's inputs without using our approach on the compositions of services.* **After** *stands for the number of parameters requiring end-users to enter information after using our approach on the compositions of services. i stands for the i_{th} recruited end-user, $1 \le i \le n$, and n is the total number of recruited end-users.*

$$Avg-effectiveness = \frac{\sum_{i=1}^{n} Effectiveness_i}{\#\ \textit{of Recruited End-users}} \tag{12}$$

TABLE 4: Average Effectiveness of Evaluating Our Approach for Reducing the Number of Input Parameters.

Domain	WSDL and REST	Web Forms
Travel	38%	62%
E-commerce	40%	46%
Finance	25%	42%
Entertainment	28%	50%

Results. Table 4 shows the results of our approach. Our approach reduces on average 50% and 32.75% of the number of input parameters on web forms and WSDL& REST respectively. Our approach performs better on web forms, because the number of parameters which can be linked together from the web forms within an application domain is greater than that of parameters which can be linked together from WSDL & RESTs. For example, a user is planning a trip; he or she has to visit a web form to search for cruises and another web form for booking cars, a large portion of the input parameters from both web forms can be linked, such as *departure* and *return date*. Using our approach, the user just needs input values to a pair of *departure* and *return date* within a Web form.

RQ4. Is our proposed approach effective in identifying similar end-users?

Motivation. Similar end-users may enter the same value to an similar input parameter of the same or different services. Reusing user inputs among end-users can help an end-user fill in a value to input parameters especially when the end-user has few historical inputs (*e.g.*, no previous inputs at all) or no proper values for the input parameters. Identifying similar end-users is the first step to reuse user inputs among multiple end-users. In this research question, we evaluate the effectiveness of our proposed approach for identifying similar end-users.

Approach. To answer the research question, we apply our approach of identifying similar end-users (*i.e.*, discussed in Section 3.4) on the six collected user profiles of the subjects to identify similar end-users. For each recruited subject, our approach recognizes two similar subjects from the other five subjects by calculating the similarity between subject activities. Moreover, we rank the identified two subjects based on their similarity values from the highest to the lowest. We use the labeled 6 subjects described in Section 4.1 as gold standard to verify the results of our approach. We use Equation (13) to calculate the effectiveness of our approach on each recruited end-user. The value of the Equation (13) is 1 or 0, because each subject only has one labeled similar subject. Finally, we use Equation (14) to measure the overall performance of our approach on all of the six subjects.

$$\text{k-}Effectiveness_i = \#\ \textit{of Labeled End-users in Top-k Results} \tag{13}$$

$$\text{Avg-k-effectiveness} = \frac{\sum_{i=1}^{n} \text{k-}Effectiveness_i}{\#\ \textit{of Recruited End-users}} \tag{14}$$

Where k is the number of recommended end-users from our approach by detecting the labeled similar subject for a subject. i stands for the i_{th} recruited subject, $1 \le i \le n$, and n is the total number of recruited subjects.

TABLE 5: Average Effectiveness of Evaluating Our Approach for Identifying Similar End-users with Different Values of k.

Top-k	Avg-Effectiveness
Top-1	66.6%
Top-2	83.3%

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

Results. Table 5 shows that our approach can effectively identify similar end-users for an end user using end-user's activities. In 83% of the cases, the top two user profiles recommended by our approach contain the most similar end-user which is manually labeled in Section 4.1. Identifying the most similar end-users to an end-user can help our approach collect suitable user inputs in order to propagate the user inputs from one end-user to another. Our approach is designed to discover the labeled subject profile with a minimum recommended similar end-user profiles.

RQ5. Can the user inputs from similar end-users help reduce the number of inputs an end-user is required to enter?

Motivation. In **RQ3**, we show that our approach only using previous user inputs from the same end-user can help reduce on average 41.38% of the number of inputs which need him or her to provide. In this research question, we evaluate the effectiveness of our overall approach to use the previous user inputs of an end-user as well as the similar end-users for reducing the number of inputs requiring the end-user to enter.

Approach. To answer the research question, we conduct two experiments:

Experiment One. Unlike the approach used in **RQ3**, we apply our overall approach to include the steps of *Identifying Similar End-users* and *Collecting User Inputs from Similar End-users* for the service compositions composed in **RQ3**. For each recruited subject, our approach collects user inputs from the recommended similar end-users in the Top 2 similar end-users as discussed in **RQ4**. Our approach uses the previous user inputs of the recruited subject and his or her similar peers to pre-fill the input parameters.

Experiment Two. For each recruited subject, our approach also collects user inputs from the human-labeled similar subject of each recruited subject. Then, we use the previous user inputs from the human-labeled similar subject to pre-fill the input parameters.

For both experiments, each recruited user verifies the pre-filled values of our approach. We use Equation (11) to calculate the effectiveness of our approach for each recruited user. The Equation (11) calculates the percentage of repetitive typing that our approach can help end-users save. Then, we use Equation (12) to measure the average performance of our approach for six recruited subjects on reducing number of input parameters.

Results. Table 6 and Table 7 show that the user inputs from similar end-users of an end-user can help improve the input parameter pre-filling. The percentage in parentheses in both Table 6 and Table 7 expresses the improvement of our approach using the previous inputs of an end-user and his or her similar peers over our approach only using the previous inputs of the same end-user (*i.e.*, in **RQ3**).

We further investigate the results of our experiments. We find that most of the improvement is resulted from the correctly filled input parameters requiring user inputs related to user preference settings, such as input parameters in a shopping search criteria page of an E-commerce website. Is is beneficial for an end-user to include the user inputs of his or her similar end-users in the process of parameter filling, when his or her previous inputs are not sufficient to fill in input parameters.

Comparing the results in Table 7 and Table 6, we find that our approach using the user inputs from the human-labeled subject to pre-fill input parameters in *Experiment Two* improves the results of our approach in *Experiment One* by 1% on WSDL and REST in e-commerce, 2% on Web Forms in e-commerce, and 2% on Web Forms in entertainment. The improvement confirms the importance of the most similar end-user discovery in the process of leveraging and using user inputs of similar end-users for an end-user's input parameter filling.

5 Threats to Validity

This section discusses the threats to validity of our study following the guidelines for case study research [20].

Construct validity threats concern the relation between theory and observation. In this paper, the construct validity threats are mainly from extracting input parameters from RESTful services and web applications. Extracting information from web pages is a challenging task. For example, the positions of labels in web forms can be various. We adopt the approach in [21] to extract information from web pages.

Internal validity threats concern our selection of subject systems, tools, and analysis method. The main threats are from manually labeling subject profiles in the section of Experiment Setup (*i.e.*, Section 4.1). We set guidelines before we conduct manual labeling process. We paid attention not to violate any guidelines to avoid the big fluctuation of results with the change of the experiment conductor. Furthermore, it is very common to include manual processes in research studies, such as the manual process in [5].

6 Related Work

In this section, we summarize the related work on assigning values to services and web forms, identifying input-output data flow of services and contexts models for web services.

Several industrial tools have been developed to help end-users fill in the web forms. Web form Auto-filling tools such as Google Chrome Autofill forms [4] and Mozilla Firefox Add-on Autofill Forms [15] help users fill in forms. RoboForm [22] and LastPass [23] are specialized in password management and provides form auto-filling function. 1Password [24] is a password manager integrating directly into web browsers to automatically log the user into websites and fill in web forms. These tools store user's information in central space, and automatically fills in the fields with the saved credentials once the user revisit the page.

Some academic studies such as [5] [25] [26] [27] propose several approaches to help end-users pre-fill web forms. Araujo *et al.* [5] mine the semantical concepts of the web form fields and use the relations of concepts to fill end-user's profile values into web forms. Hartman *et al.* [25] propose a context-aware approach using user's contextual information to fill in web forms. They propose a novel mapping process for linking contextual information and user interface components of web forms. Toda *et al.* [26] propose a probabilistic approach using the information extracted from data-rich text as the input source to fill values into web forms. Wang *et al.* [27] propose an approach identifying

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

TABLE 6: Average Effectiveness of Evaluating Our Approach Using Similar End-users' Inputs for Reducing the Number of Input Parameters in Experiment One.

Domain	WSDL and REST (Improvement)	Web Forms (Improvement)
Travel	38% (+0%)	66% (+4%)
E-commerce	46% (+6%)	54% (+8%)
Finance	31% (+6%)	46% (+4%)
Entertainment	32% (+4%)	52% (+2%)

TABLE 7: Average Effectiveness of Evaluating Our Approach Using Similar End-users' Inputs for Reducing the Number of Input Parameters in Experiment Two.

Domain	WSDL and REST (Improvement)	Web Forms (Improvement)
Travel	38% (+0%)	66% (+4%)
E-commerce	47% (+7%)	56% (+10%)
Finance	31% (+6%)	46% (+4%)
Entertainment	32% (+4%)	56% (+4%)

and using the relations between similar input parameters to fill values into web forms. Academic researchers propose approaches for assigning values to parameters of services. AbuJarour *et al.* [6] propose an approach to generate annotations for web services by sampling their automatic invocations. They use four sources, such as random values, to automatically assign values for input parameters of web services. Firmenich *et al.* [28] analyze user's interactions with web forms and enhance web forms to assist users to fill in web forms. Wang *et al.* [29] propose a classification of user inputs and use the classification to help pre-fill web forms using an end-user's previous inputs.

The approaches [30] [31] [32] of submitting web forms automatically in deep web crawling also requires similar web forms linkage and automatic assignment of values to web forms. Kabisch *et al.* [30] propose VisQI a deep web integration system to transform web interfaces into hierarchically representations classified into different domains for linking different interfaces. Kriegel *et al.* [31] introduce an approach to classify similar websites as sets of feature vectors. Nguyen *et al.* [32] propose an approach using learning classifiers to extract element labels from web form interfaces. Mapping the labels to elements correctly is the key step to link similar web interfaces.

All of the above approaches treat services individually and do not consider the value assignment of input parameters in the context of service composition.

The approaches from research studies such as Thomas *et al.* [7], Gerede *et al.* [8] and Li *et al.* [33] identify input-output data flow for chaining services to help end-users complete their tasks. They do not take into consideration the propagation of user inputs across services and only use the name of input and output parameters for chaining services.

Mrissa *et al.* [34] propose a context model for composing services. Blake *et al.* [35] propose a context model containing user's contextual information for identifying relevant services to end-users. However these proposed context models are not designed for propagating user inputs across services and assigning values to services.

7 Conclusion

Repetitively entering the same information into composed services causes unnecessary interruptions and decreases the efficiency of service execution. In this study, we propose an approach to prevent end-users from repetitive data entry tasks. Our approach propagates user inputs to different services among multiple end-users by identifying similar parameters and linking end-users who perform similar tasks. We propose a context-aware Meta-data Model to store an end-user's previous inputs. Our approach pre-fills the input parameters using the previous inputs from end-users. Our approach discovers similar end-users by calculating the similarity between the previous activities of end-users. Our approach also leverages and consumes the user inputs of the similar end-users to an end-user for pre-filling input parameter.

To identify a proper value for an input parameter, our approach achieves an effectiveness of 86.5% on average and outperforms the baseline approach. We also evaluate the effectiveness of pre-filling user inputs into parameters. Our results show that on average 78% of the input parameters are filled correctly and our approach outperforms the baseline approach by filling 23% more parameters on average. Furthermore, our overall approach can reduce on average 41% of input parameters requiring an end-user to enter values for composed services using the previous user inputs of a single end-user. With the addition of user inputs from the similar end-users to the end-user, our approach can reduce on average 46.5% of input parameters. In the future, we plan to implement a working prototype of our approach and recruit more end-users to conduct user study on our system.

Acknowledgments

We would like to thank Ms. Diana Lau at IBM Toronto CAS research for her valuable feedback to this research. This research is partially supported by IBM Canada Centers for Advance Studies.

References

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

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

[2] B. Upadhyaya, R. Tan and Y. Zou, *An Approach for mining service composition patterns from execution logs*, Journal of Software: Evolution and Process, Volume 25, Number 8, pp. 841-870, Wiley, August 2013.

[3] E. Rukzio, C. Noda, A De Luca, J. Hamard, and F. Coskun. Automatic form filling on mobile devices. Pervasive Mobile Computing, vol. 4, no. 2, pp. 161-181, 2008.

[4] Autofill forms - Google Chrome. Available at *https://support.google.com/chrome/answer/142893*. Last Accessed on Jan 19, 2014.

[5] S. Araujo, Q. Gao, E. Leonardi, J. Houben. Carbon: domain-independent automatic web form filling. In Proceedings of the 10th International Conference on Web Engineering (ICWE'10), Berlin, Heidelberg, pp. 292-306, 2010.

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

[7] J. P. Thomas, M. Thomas, and G.Ghinea, *Modeling of web services flow*, IEEE International Conference on E-commerce, pp. 391-398 June 24-27, 2003.

[8] C.E. Gerede, R. Hull, O.H. Ibarra, and J. Su, *Automated composition of e-services: lookaheads*, Proceedings of 2004 International Conference on Service Oriented Computing, pp. 252- 262, New York City, USA, Nov 15-18, 2004.

[9] S. Wang, B. Upadhyaya, I. Keivanloo, Y. Zou, J. Ng and T. Ng, *Automatic Propagation of User Inputs in Service Composition for End-users*, Proceedings of the 21th IEEE International Conference on Web Services (ICWS), June 27 - July 2, 2014, Alaska, USA. IEEE

[10] R. T. Fielding, R. N. Taylor, *Principled design of the modern Web architecture*, Journal of ACM Transactions on Internet Technology, Volume 2 Issue 2, Pages 115-150, May 2000.

[11] WSDL. *http://www.w3.org/TR/wsdl*. Last Accessed on Jan 14, 2014.

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

[13] Sun Microsystems, *Web Application Description Language: W3C Member Submission 31*, World Wide Web Consortium, 2012.

[14] B. Upadhyaya, Y. Zou, S. Wang and J. Ng, *Automatically Composing Services by Mining Process Knowledge from the Web*, Proceedings of 11th International Conference on Service Oriented Computing, pp. 267-282, Berlin, Dec 2-5, 2013.

[15] Autofill Forms - Mozilla Firefox add-on. Available at *https://addons.mozilla.org/en-US/firefox/addon/autofill-forms/?src=ss*. Last Accessed on Jan 19, 2014.

[16] D. Flanagan, *JavaScript: The Definitive Guide*, O'Reilly & Associates. pp. 312C313. ISBN 0-596-10199-6.

[17] WordNet. *http://wordnet.princeton.edu/*. Last Accessed on Jan 14, 2014.

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

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

[20] R.K.Yin, *Case Study Research: Design and Methods*-Third Edition, 3rd. SAGE Publications, 2002.

[21] B. Upadhyaya, F. Khomh, Y. Zou, *Extracting RESTful Services from Web Applications*, Proceedings of 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pp. 1-4, Taipei, Dec 17-19, 2012.

[22] RoboForm: *http*://www.roboform.com/. Last Accessed on April 15, 2014.

[23] LastPass: *http*://www.lastpass.com/. Last Accessed on April 15, 2014.

[24] 1Password: *https*://agilebits.com/onepassword. Last Accessed on October 30th, 2014.

[25] M. Hartman and M. Muhlhauser, *Context-Aware Form Filling for Web Applications*, ICSC' 09. IEEE International Conference on Semantic Computing, 2009, pp. 221-228.

[26] 5. G. Toda, E. Cortez, A. Silva and E. Moura, *A Probabilistic Approach for Automatically Filling Form-Based Web Interfaces*, The 37th International Conference on Very Large Data Base, August 29th - September 3rd 2011, Seattle, Washington.

[27] S. Wang, Y.Zou, B. Upadhyaya, and J.Ng, *An Intelligent Framework for Auto-filling Web Forms from Different Web Applications*, 1st International Workshop on Personalized Web Tasking, co-located with IEEE 20th ICWS, June 27, 2013, Santa Clara Marriott, California, USA.

[28] S. Firmenich, V. Gaits, S. Gordillo, G. Rossi and M. Winckler, *Supporting Users Tasks with Personal Information Management and Web Forms Augmentation*, Proceedings of 12th International Conference (ICWE), pp. 268-282, Berlin, Germany, July 23-27, 2012

[29] S. Wang, Y. Zou, B. Upadhyaya, I. Keivanloo and J. Ng, *An Empirical Study on Categorizing User Input Parameters for User Inputs Reuse*, Proceedings of the 14th International Conference on Web Engineering (ICWE 2014), Springer, July 1st - 4th, 2014, Toulouse, France.

[30] T. Kabisch, E. C. Dragut, C. Yu, and U. Leser, *Deep web integration with VisQI*, Proceedings of VLDB Endow., vol. 3, pp. 1613-1616, September 2010.

[31] H. P. Kriegel and M. Schubert, *Classification of Websites as Sets of Feature Vectors*,Proceedings of Databases and Applications, pp.127-132,2004

[32] H. Nguyen, T. Nguyen, and J. Freire, *Learning to Extract form Labels*, Proceedings of VLDB Endow., vol. 1, pp. 684-694, August 2008.

[33] L. Li, and W. Chou, *Automatic Message Flow Analyses for Web Services Based on WSDL*, Proceedings of 2007 IEEE International Conference on Web Services, pp. 880-887, Salt Lake City, UT, USA, July 9-13, 2007.

[34] 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 Techn. 8(1) 2007.

[35] 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.

Shaohua Wang is currently a Ph.D student, under the supervision of Professor Ying Zou, in the School of Computing at Queen's University. He is a research student of IBM Centers for Advanced Studies, IBM Canada. He holds IBM Ph.D Fellowship Award. His research interests include: service-oriented computing, web mining, software maintenance and evolution, empirical software engineering and artificial intelligence. He has published several papers in international conferences and journals, including ICSOC, ICWS, ICWE, MSR and EMSE.

Ying Zou is a Canada Research Chair in Software Evolution. She is an associate professor in the Department of Electrical and Computer Engineering and cross-appointed to the School of Computing at Queen's University in Canada. She is a visiting scientist of IBM Centers for Advanced Studies, IBM Canada. Her research interests include software engineering, software re-engineering, software reverse engineering, software maintenance, and service-oriented architecture.

Iman Keivanloo is currently a Post Doctoral Fellow with Dr. Ying Zou at Software Re-engineering Research Group in the Department of Electrical and Computer Engineering at Queen's University. His research interests are (1) source code similarity search, (2) clone detection, (3) source code recommendation/completion, and (4) their applications for software evolution and maintenance. He also interested in applications of Linked Data and Semantic Web in software engineering, development, and maintenance.

Bipin Upadhyaya is currently a Post Doctoral Fellow with Dr. Ying Zou at Software Re-engineering Research Group in the Department of Electrical and Computer Engineering at Queen's University. His research interests includes: service-oriented architecture and software engineering.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSC.2014.2378278

Joanna Ng is currently the Head of Research at IBM Canada Software Laboratories, Center for Advanced Studies. She is also a Senior Technical Staff Member of IBM Software Group. In her current position, she directs the organization to conduct industrial research in collaboration with academic researchers, IBM researchers and software technologists.

Tinny Ng is currently a Research Staff Member for IBM CAS Research Canada at IBM Canada Lab. Her primary focus is to manage research portfolios and transfer advanced research into strategic product solutions for IBM Software Group. Previously, Tinny was a scenario architect and a tool lead of IBM Software Group Strategy and Technology focusing on cross-brand integration capability and consumability.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.