Automatic Propagation of User Inputs in Service Composition for End-users

Shaohua Wang, Bipin Upadhyaya, Ying Zou, Iman Keivanloo
Queen's University
Kingston, Ontario, Canada
e-mail: shaohua@cs.queensu.ca, {bipin, iman.keivanloo, ying.zou}@queensu.ca

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

***Abstract*—End-users conduct various on-line activities. Quite often they need re-visit websites and use on-line services to perform repeated activities, such as on-line shopping. The end-users are required to enter the same information into various web services to accomplish such repeated tasks. Typing redundant information repetitively into such services unnecessarily interrupts end-users and negatively impacts the user experience. In this study, we propose an approach to prevent end-users from unnecessary interruption caused by redundant information typing. Our approach propagates user inputs across services by linking similar input and output parameters. Our approach also pre-fills values to the input parameters which could not be filled by the values from other input or output parameters. We propose a meta-data model for storing user inputs and an Input Parameter Context Model for identifying similar input or output parameters. We also summarize four scenarios of providing values to an input parameter. We have implemented our approach and evaluated it on the real world services through an empirical study. Our overall approach can reduce on average 37% of input parameters through execution of a composed service.**

***Keywords*-Information Reuse; Service Composition; Input Parameter Pre-filling**

I. Introduction

Nowadays, web is playing an important role in people's daily life. On-line end-users can conduct various tasks, such as booking airline tickets and on-line shopping. An end-user re-visits websites or on-line services and compose them to perform on-line tasks to meet his or her needs [2][3]. For example, 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 are linked implicitly by a user. To invoke each service, the end-user needs provide values to input parameters of services. However the current practice of assigning values to input parameters of services during a composition brings end-users the following two challenges:

- **Limitation in Propagation of end-user inputs across services.** Among the parameters of composed services, the input and output parameters semantically related or identical should be linked together, because the information can be propagated among these related parameters. For example, both services *ticket liquidator* and *Greyhound* require the end-user to input the quantity of tickets he or she wants to buy. The two input parameters requiring the number of quantity from two services should be linked. If one of services is filled with a quantity value, the value should be propagated to the other one.
- **Lack of approaches for pre-filling services of a composition.** The end-users are often required to provide values to the input parameters of composed services. Usually the information provided by the end-users is redundant and repetitive, because these information could be entered into other services previously and not recorded. For example, both services *ticket liquidator* and *Greyhound* require the end-user's name for buying tickets. E.Rukzio et al. [1] found that users are four times faster on a smart phone when they just have to correct pre-filled form entries compared to entering the information from scratch. Pre-filling input parameters of services can improve the efficiency of service invocation.

There is an urgent need to call for an approach that prevents the end-users from interruptions caused by unnecessarily inputting values to services during the service composition.

Recently, several approaches have been developed to fill in Web forms. Google Chrome Autofill forms [11] fill in web forms using end-user's previous inputs. Araujo *et al.* [12] propose a concept-based approach for automatic web form filling. These approaches treat input parameters of services individually and do not identify the relations among parameters during the course of assigning values to input parameters. AbuJarour *et al.* [5] propose an approach to assign values for input parameters of web services automatically. Thomas et al. [21] and Gerede et al. [22] identify input-output data flow for chaining services. However all the above approaches heavily rely on the names of parameters for identifying similar parameters. When the names of parameters are not available for similarity calculation because of bad naming such as incorrect spelling, these approaches do not perform well. All the above approaches are not designed for filling values to input parameters under the context of service composition.

In this paper, we propose an approach that captures user's previous inputs, propagates the user inputs across different services, and pre-fills input parameters of services during the service composition. We aim to maximize the reuse of end-user's inputs and take our best effort to prevent end-users from

1. $http://www.ticketliquidator.com/default.aspx$
2. $http://www.greyhound.ca/$

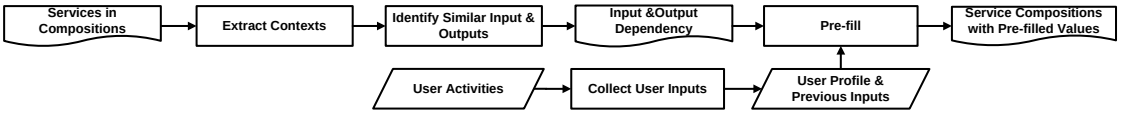

Fig. 1: Overall steps for propagating the inputs of end-users across different services during service composition.

being interrupted by typing redundant information during the service execution. The major contributions of our research are listed as follows:

- We propose an Input Parameter Context Model for identifying similar parameters of services. Each parameter has three contexts: Textual Context, Task Context and Neighbors Context in our model. Our approach uses the contexts of parameters for identifying similar parameters to enable the reuse of user's inputs across services.
- We propose a meta-data model for storing an end-user's previous input. Our meta-data model is designed to store more information of a user input. Our approach of pre-filling uses the model for context-aware matching between the end-user inputs and the input parameters during the service composition.
- We test the effectiveness of our proposed approach through three experiments and compare it with baseline approaches we build. The first two experiments measure the effectiveness of two steps: identifying similar parameters and pre-filling values in our overall approach. The last experiment measures the effectiveness of our overall approach on reducing the number of input parameters requiring end-user's inputs. Our approach considerably save end-users from providing values to on average 37% of input parameters of services of a composition.

The rest of this paper is organized as follows. Section II presents the background of this study. Section III introduces the proposed approach. Section IV introduces the case study on our approach. Section V summarizes the related literature. Finally, Section VI concludes the paper and outlines some avenues for future work.

II. Background

Our paper is primarily focused on reusing end-user's inputs among the web services during the service composition. A web service is a software module designed to help inter-operation between machines over the web.

There are two types of protocol specifications for exchanging information in the implementation of Web services, namely Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) [6]. Web Services Description language (WSDL) [7] is an XML based interface description language used for describing the functionality of a web service, and often used in combination with SOAP. RESTful services [8] simplify the development, deployment and invocation of web services. The end-users conduct various tasks through web applications (*e.g.*, web forms). In this paper, we consider a web application as a third type of on-line services.

In a service invocation chain, there are three types of linkages [4]: 1) User-to-Service: A user invoking a service with input parameters is User-to-Service. 2) Service-to-Service: A Service-to-Service linkage occurs when a service invokes another service. 3) Service-to-User: The service needs human inputs or confirmation is Service-to-User. In this study, our approach is to facilitate these three linkages.

III. Overview of Our Approach

In this paper, we propose an approach that propagates user inputs across different services and pre-fills the input parameters of services using end-user's previous inputs during the service compositions. Figure 1 shows the steps of our approach. Our approach consists of three major steps:

1) Collecting and storing user inputs. The end-user's previous inputs can be an excellent source of feeding input parameters, because the end-users could perform the similar services which consume the previous inputs.
2) Collecting contexts of parameters and building input-output dependency. We extract the contexts of parameters for identifying similar parameters. We build input-output dependency of parameters using the extracted contexts for propagating user's inputs across different services. The propagation of user inputs can reduce the number of input parameters requiring user's inputs.
3) Pre-filling values to operations of services. We use the end-user's personal data and previous inputs to fill the input parameters requiring a value from end-users.

A. Four Scenarios of Assigning Values to Input Parameters

We identify four scenarios of supplying values to an input parameter within a service of service compositions. The four scenarios are listed as follows:

1) *Scenario 1.* Sharing values to the similar input parameters of operations. This scenario has two versions: 1) The operations are from one service. 2) The operations are from different services. If two input parameters from different operations are semantically related, the value provided to one of them should be propagated and used to fill the other one.
2) *Scenario 2.* Reusing values of output parameters to fill in an input parameter. This scenario has two versions: 1) The output parameters and the targeted input parameter are from different operations but one service. 2) The output parameters and the targeted input parameter are from different operations that are from different services. The output of an output parameter can be used to fill an input parameter. For example, an operation of a service generates a city name by using zip code, and an input parameter of a weather forecast service requires a city name to give weather forecast.
3) *Scenario 3.* Using external data sources to assign values to input parameters. The personal profile data and previous inputs from end-users are used as the external data sources to fill in an input parameter. The end-users could perform the similar services consuming the previous inputs or information in personal profile.

4) *Scenario 4.* Requiring typing from End-users. If the aforementioned sources are not available for an input parameter, the end-users need input 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 four scenarios.

B. 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 [10][11][12] 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 pair model lacks of information for conducting context-aware matching between the values and input parameters. For example, there are several values for a key "contact phone number", the end-users use one value for flight booking and another value for hotel reservation. Besides storing the type of the information, we modify and enrich the key-value pair model by attaching more information to the piece of information and propose a Meta-Data Model for storing user inputs.

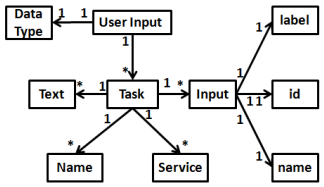

Fig. 2: Description of Meta-Data Model

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

Meta-Data Model of User Input: Figure 2 shows the description of our proposed meta-data model. Figure 3 shows an example of storing an email address "shaohua@cs.queensu.ca" entered into the input field of 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 is an operation in WSDL services, or a resource and its associated action if applicable (*e.g.*, get or delete) or a web form of a web application. The data types are String, Numeric, Boolean and Date. 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. The task property describes the applicable tasks for 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 in WSDL, the resource name and its associated actions of a RESTful service, and a label of a web form or a HTML page name of a web application. As shown in Figure 3, the name property of the task of the email "shaohua@cs.queensu.ca" is the label "Sign in" of the web form.
- ***Text Property***: defines the description of a task. To retrieve a description for a task, we extract the description of an operation in WSDL, the descriptive text of a resource in a RESTful service, and text information of a web form (*e.g.*, web form name) as a ***text***. If the description is not available, we assign NULL to this property. In the example of Figure 3, the text property of the task "Sign in" of the email "shaohua@cs.queensu.ca" is the web form name "SignInform".
- ***Service Property***: records the name of a service in which the task is performed. We use the name of the WSDL or 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 and assign NULL to id and name.

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

In this study, 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 possible user's inputs (*e.g.*, user id), and generates a log. 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 from Sahi log 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. The end-users can run our tool and surf the web through web browsers as usual.

3. *www.ebay.com*
4. http://www.soapui.org/
5. http://sahi.co.in/

C. Collecting Contexts of Parameters and Building Input-Output Dependency

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

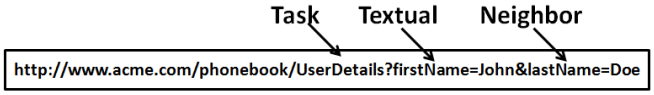

Fig. 4: An example annotated screenshot of showing the four contexts of a parameter from a RESTful service.

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 shows an annotated screenshot of showing contexts of a parameter from a RESTful service. The three contexts of a parameter are listed as follows:

- **Textual Context** (denoted as P^{Text}): defines the textual information of the parameter. In most cases, the value of the textual information is the name of a parameter from an operation 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[6] 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* of the HTML Dom [15] element defining the input parameter in user interface.
- **Task Context** (denoted as P^{Task}): stores the task information the parameter is used for. A task is an operation in WSDL services, or a resource and its associated action if applicable (*e.g.*, get or delete) or a web form of web applications. We store the descriptive information and a name for the task information. We extract the description if applicable and the name of an operation in defined in a WSDL file. We extract the resource name and its associated actions if applicable, and the description of the resources if applicable for a RESTful service. For a web application, we extract the task information from page name or a web form name. For example, the task name of the input parameter firstName in Figure 4 is "UserDetails" and the descriptive information can be obtained from the web page introducing this operation.
- **Neighbors Context** (denoted as $P^{Neighbor}$): stores a group of other parameters required by the same operation. For example, the neighbor of the input parameter firstName in Figure 4 is "lastName".

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

To identify meaningful words from the contexts, we conduct word normalization on the contexts extracted from services to calculate the similarity between parameters in four steps. First, we decompose any possible compound words. 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 No. (*e.g.*, City1), Underscore separator (*e.g.*, departure_city) and Dash separator (*e.g.*, Find-city). Next, we use wordNet [13] a lexical database organizing the English words into a set of synonyms to remove non-English words. Third, we remove the possible stop words using a pre-defined list of stop words such as "the". Finally, we use porter stemmer [14] to reduce derived words to their stem, base, or root form (*e.g.*, "reserves" and "reserved" both get mapped to "reserve").

After the word normalization, we conduct the context similarity calculation. A context such as task information P^{Task} of a parameter could contain more than one word, for example a task name "book car". We formulate a context into a phrase, Ph, which has 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 words. When calculating the semantic similarity between two phrases Ph_i and Ph_j, we use the following steps:

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 the words are same, then Ph_i=Ph_j.
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 have the same words in Ph_i, the Ph_j is *contained* by Ph_i, we use the 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.

3) If there is no *containment* relation between Ph_i and Ph_j, we use the following approach to calculate the semantic similarity between two phrases.
   - Step 1: calculate the number of pairs of common words of Ph_i and Ph_j.
   - Step 2: calculate the semantic similarity between every pair of words (*i.e.*, sim(W_a, W_b) denotes the similarity value between two words W_a and W_b) from two phrases after excluding the common words from calculation, and then sum up all the similarity values, which is denoted as $Sum_{sim} = \sum_{W_a \subset Ph_i} \sum_{W_b \subset Ph_j}$ sim(W_a, W_b)
   - Step 3: we use the Equation (2) to calculate the

6. http://www.w3.org/Submission/wadl/

similarity between two phrases.

$$sim(Ph_i, Ph_j) = \frac{|Ph_i \bigcap Ph_j| + Sum_{sim}}{|Ph_i \bigcap Ph_j| + |pairs\ of\ words|} \quad (2)$$

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

$$sim(P_i, P_j) = A * sim(P_i^{text}, P_j^{text}) + B * sim(P_i^{task}, P_j^{task}) + C * sim(P_i^{neighbor}, P_j^{neighbor}) \quad (3)$$

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 and A is its weight. $sim(P_i^{task}, P_j^{task})$ defines the semantical similarity between the context of Task Information of two parameters and B is its weight. $sim(P_i^{neighbor}, P_j^{neighbor})$ defines the semantic similarity between the context of Group of Neighbors of two parameters and C is its weight. We assign weights of 0.33 to each context. We determined experimentally that these weights generate reasonable results.

Given a set of services $S_1, S_2, \ldots, S_n$ (n is the number of services already in the composition) in an on-the-fly service composition (*i.e.*, the end-users could add new services into the composition, and S_1 is the first service selected into composition), we extract all the input and output parameters of services, then calculate the similarity values between each input parameter and all the other parameters using their contexts, so each input parameter has a vector of similarity values. We remove all the similarity values which are less than a threshold value (*i.e.*, an empirically derived threshold).

To fill in an input parameter of an operation from the service S_i, $1 \leq i \leq n$, the parameters of operations from service the S_j, $i<j \leq n$ cannot be used as data sources for this input parameter from the service S_i, because these parameters are from subsequent services. We remove such parameters from our analysis when we identify the similar parameters for this input parameter. We rank the similarity values from highest to lowest within the vector for each input parameter. We build input-output dependency graph based on the vectors of similarity values among parameters.

If an input parameter of an operation can be filled by the input of an input parameter of an operation from the same service that this input parameter is in or from preceding services (*i.e.*, **Scenario 1** in Section III-A), this input parameter does not need an input from end-users. Furthermore, if an input parameter of an operation can be filled by an output parameter from a service that this input parameter is in or from the preceding services (*i.e.*, **Scenario 2** in Section III-A), this input parameter does not need an input from end-users. Finnaly, using the input-output dependency graph, we reduce the number of input parameters requiring user inputs by identifying the relations between parameters in **Scenario 1** and **Scenario 2**.

D. Pre-filling Values to Input Parammeters of Services

We fill in the input parameters which need inputs from end-users by using the previous inputs to other services from end-users (*i.e.*, **Scenario 3** of providing values to input parameters in Section III-A). To fill in an input parameter P, we identify the proper previous user inputs stored in the Meta-Data Model proposed in Section 2. We traverse all the user inputs and calcualte the similarity value between each user input and the input parameter P. Then we select the user input with the highest similarity value as the input for P.

Given a user input I and an input parameter P, we calculate the semantic similarity between them in the followig steps:

- Step 1: We use the approach in Section III-C to conduct word normailization on all the properties of the input I.
- Step 2: We traverse all the possible ***Task Properties*** $\{I^{Task_1}, I^{Task_2}, \ldots I^{Task_n}\}$ of the input I to verify whether this input can be used for the input parameter, n is the number of Tasks the user input used for. We calculate the similarity between a task I^{Task_i} $(1 \leq i \leq n)$ and the input parameter P in the following steps:
  1) We merge all the words from the three properties (*i.e.*, Label, ID and Name) of ***Input Property*** of I^{Task_i} as the input information of I^{Task_i}, denoted as $I^{Task_i^{In}}$. We use Equation (1) proposed in Section III-C to calculate the similarity value between $I^{Task_i^{In}}$ and the ***Text Context*** P^{Text} of P, which is denoted as sim($I^{Task_i^{In}}$,P^{Text}).
  2) We merge all the words from Text and Name properties of I^{Task} as Textual information, which is denoted as $I^{Task_i^{Text}}$. We use the Equation (1) proposed in Section III-C to calculate the similarity value between $I^{Task_i^{Text}}$ and the ***Task context*** P^{Task} of P, denoted as sim($I^{Task_i^{Text}}$,P^{Task}).
  3) We use Equation (4) to calculate the similarity between a ***Task Property*** of input I and an input parameter P.

$$sim(I^{Task_i}, P) = M * sim(I^{Task_i^{In}}, P^{Text}) + N * sim(I^{Task_i^{Text}}, P^{Task}) \quad (4)$$

*where $sim(I^{Task_i}, P)$ defines the similarity between the **Task Property** $Task_i$ of the user input and the parameter.*

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

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, because these weights generate reasonable results experimentally.

- Step 3: We select a $Task_j$ $(1 \leq j \leq n)$ with the highest similarity value with the input parameter as the match for the input parameter.

IV. Case Study

We introduce our dataset and the research questions. For each question, we present the movitaion of the question, the analysis approach and the results of each question.

A. Case Study Setup

We conduct our study on three types of services: SOAP-based, RESTful and web applications. We consider SOAP-based and RESTful services as the services with service descriptions and web forms of web applications as the services without service descriptions. In our dataset, the SOAP-based services are all in WSDL; the RESTful services are described in Web pages.

TABLE I: Statistics About Our Collection 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

In total, we collect and download 640 public available WSDL files. We use programmableWeb[7] to collect 60 URLs of RESTful services and download the web pages containing the APIs. We manually collect the information related to RESTful services from web pages for input parameter contexts extraction, such as the description of resources. The 700 services with service descriptions fall into 4 domains: Travel(*e.g.*, book flights), Ecommerce (*e.g.*, buying shoes), Finance (*e.g.*, check a stock price) and Entertainment(*e.g.*, search a TV show). We use google to search for websites for each domain to download web forms. We choose websites listed on the top of the Google ranking. In total we download 50 Web forms. Table I describes the descriptive staticstics of the collected services.

To test our idea of pre-filling input parameters using the user previous inputs, we collect user inputs through our input collecter proposed in III-B. The first author of this paper used the tool to track his inputs on web forms requiring his basic information such as name, email, address, phone number. Furthermore, we use different pieces of information of one type for diverse tasks, for example, we use different email addresses for various tasks such as login, contact information.

B. Research Questions

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

RQ 1. Can the proposed Input Parameter Context Model 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 parameterss using our proposed Input Parameter Context Model (IPCM).

Approach. we build a baseline approach only using names of parameters for identifying similar parameters to compare with our approach of identifying similar parameters. We collect all the input and output parameters from 700 WSDL and RESTful services, and 60 web forms seperately. We calculate the similarity values between each input parameter and the other parameters using our approach and the baseline approach. Each approach forms pairs of parameters and each pair has an input parameter and its most similar parameter (*i.e.*, the parameter has the highest similarity value with the input parameter). After collecting pairs of parameters by using two appraoches, we sample pairs of parameters with confidence level 95% [24] for manual verification of the results of our approach and baseline approach, we randomly sample 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 the Equation (5) to measure the effectiveness.

$$Effectiveness = \frac{Number\ of\ Correct\ Pairs}{Number\ of\ Input\ Parameters} \quad (5)$$

TABLE II: Results of the Evaluations of Our Approach and Baseline Approach of Identifying Value Sources for Input Parameters.

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

Results. Table II shows the results of our evaluation. Our approach achieves an accuracy 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 could not work.

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

Motivation. Pre-filling values to input parameters of operations of services is an essential step to save end-users from repetitive typing. We are interested in evaluating our pre-filling appraoch using the proposed Meta-data Model containing task information of user inputs and Input Parameter Context Model containing contexts of parameters.

Approach. Pre-filling an end-user's input to an input parameter requires the similarity calculation between the end-user's input and the input parameter to check wether this end-user is suitable for the input parameter or not. In this study, the end-user's inputs are described using Meta-data Model (MDM) and context information of a parameer is in the Input Parameter Context Model (IPCM). Our approach of pre-filling uses MDM and IPCM to caluculate the similarity between an end-user's input and an input parameter. To compare with our approach, we build a baseline approach based on the key-value pair model, mentioned in Section III-B, describing an end-user's input and the name of an input parameter.

To be able to validate the results manually, we sample the input parameters with confidence level 95% [24], we randomly select 376 input parameters from WSDL and RESTful services and 230 input parameters from web forms. Our approach and the baseline approach both use the collected end-users'

7. http://www.programmableweb.com/

inputs through our input collector to fill in the selected input parameters. To measure the effectiveness of our approach, we calculate precison and recall using the Equation (6) and Equation (7).

$$Precison = \frac{|Correct\ Filled\ Input\ Parameters|}{|Filled\ Input\ Parameters|} \quad (6)$$

$$Recall = \frac{Correct\ Filled\ Input\ Parameter}{|Input\ Parameters\ Need\ To\ Be\ Filled|} \quad (7)$$

TABLE III: Results of the Evaluations of Our Approach and Baseline Approach of Filling Input Parameters.

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

Results. Table III shows the results of our evaluation. Our approach outperforms the baseline approach. The recalls of the two approachs 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, phone number, however web forms are usually designed to require these basic personal information. The baseline approach cannot distinguish the different pieces of information of one type under 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 approach of pre-filling values to input parameters achieves a precision of 78% on average.

RQ3. Can our overall approach reduce the number of inputs required by end-users?

Motivation. The RQ1 and RQ2 measure the effectiveness of two different sub-steps of our overall appaoch. In this question, we evaluate the effectiveness of our overall approach for reducing the number of inputs required by end-users during the service composition.

Approach. We compose services to form compositons of services. The flows of the web forms from a web application are defined by the web application designers and the number of flows is limited. For simplicity, we just combine web forms within a web application based on the desinged flows to form service compostions. To WSDL and RESTful services, we randomly select 30 services from each domain and build input and output data flow between services, and use the flow data dependency to compose the selected services. We believe the services selected from one domain are highly related and can be composed into reasonable paths of services.

After forming the service compositions, we remove all the compositions only having one service. For each type of services (WSDL and REST, and web forms), we randomly select 5 service compositions from the generated compositions for each domain. Since there are errors in the compositions generated by the automatic aprpoach, we manually correct them to remove the possible bias of our results because of the errors of service compositions. We apply our overall approach on the randomly selected compositions of services (*i.e.*, a service composition means a scenario of composing services). Our overall approach analyzes the relationships between parameters and generates a list of input parameters which need user's inputs, then use our collected end-user inputs to fill these input parameters. We use the Equation (8) to measure the effectiveness of our approach.

$$Effectiveness = \frac{Num\ of\ Before\ -\ Num\ of\ After}{Num\ of\ Before} \quad (8)$$

where **Num of Before** *stands for the number of input parameters requiring end-user's inputs without using our approach on the compositions of services.* **Num of After** *stands for the number of parameters requring end-users to enter information after using our approach on the compositions of services.*

TABLE IV: Results of the Evaluation of Our Approach of Reducing the Number of Input Parameters.

Domain	WSDL and REST	Web Forms
Travel	34%	56%
E-commerce	36%	42%
Finance	25%	38%
Entertainment	22%	42%

Table IV shows the results of our approach. Our approach reduces on average 44.5% and 29.25% of the number of input parameters on web forms and WSDL& REST respectively. Our approach performs better on web forms, because the numebr of parameters which can be linked together from the web forms within an application damain is much greater than that of parameters which can be linked together from WSDL& RESTs. For example, a user conducts the task of planning a trip, he or she needs visit a web form of searching cruises and a web form of booking cars, a huge portion of the input parameters from two 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.

V. 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.

Web form Auto-filling tools such as Mozilla Firefox Add-on Autofill Forms [10] and Google Chrome Autofill forms [11] help users fill in forms. RoboForm [16] and LastPass [17] are specialized in password management and provides form auto-filling function. Academic researchers propose approaches for assigning values to parameters of services. AbuJarour *et al.* [5] 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. The approaches [18][19][20] of submitting web forms automatically in deep web crawling also requires automatic assignment

of values to web forms. All the 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.* [21], Gerede *et al.* [22] and Li et al. [23] 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.* [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 end-users. However these proposed context models are not designed for propagating user inputs across services and assigning values to services.

VI. Conclusion and Future Work

Repetitively entering the same information into composed services provides end-users uncessary interruptions and decreases the efficiency of service execution. In this study, we propose an approach preventing end-users from repetitively assigning values to services. Our approach propagates user inputs across differnet services by identifying proper input or output parameters as value sources for an input parameter using the proposed Input Parameter Contexts. Furthermore, we propose a Meta-data Model for storing user's previous inputs. Our approach pre-fills the input parameters using the previous inputs from end-users.

The approach of identifying input sources for input parameters achieves an accuracy of 86.5% on aeverage and ourperforms the baseline approach only using names of parameters for similarity calculation. We also evaluate the effectiveness of pre-filling user existing inputs into input parameters. Our experiment results show that on average 78% of the input parameters are filled correctly and our approach outperforms the baseline approach significantly by filling 23% more parameters on average. Furthermore, our overall approach can reduce on average 37% of input parameters of composed services. In the future, we plan to implement a working prototype of our approach and recruit end-users to conduct user case study on our system.

Acknowledgment

We would like to thank 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] 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.

[2] 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.

[3] 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.

[4] 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 (ICSOC), pp 267-282, Berlin, Germany, Dec 2-5, 2013.

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

[6] R. T. Fielding, R. N. Taylor, *Principled design of the modern Web architecture*, Journal of ACM Transactions on Internet Technology(TOIT), Volume 2 Issue 2, Pages 115-150, May 200.

[7] Web Services Description Language (WSDL). *http://en.wikipedia.org/wiki/Web_Services_Description_Language*. Last Accessed on Jan 14, 2014.

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

[9] 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.

[10] 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.

[11] Autofill forms - Google Chrome. Available at *http://support.google.com/chrome/bin/answer.py?hl=en&answer=142893*. Last Accessed on Jan 19, 2014.

[12] 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, Berlin, Heidelberg, 292-306.

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

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

[15] HTML DOM. *http://en.wikipedia.org/wiki/Document_Object_Model*. Last accessed on Jan 16, 2014.

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

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

[18] 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.

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

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

[21] J. P. Thomas, M. Thomas, and G.Ghinea, *Modeling of web services flow*, IEEE International Conference on E-commerce (CEC), June 24-27, 2003.

[22] 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 (ICSOC), pages 252- 262, New York City, USA, Nov 15-18, 2004

[23] 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.

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

[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 Techn. 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)