An Intelligent Framework for Auto-filling Web Forms from Different Web Applications

Shaohua Wang*

School of Computing, Queen's University, Kingston, Ontario, Canada
E-mail: shaohua@cs.queensu.ca, *Corresponding author

Ying Zou, Iman Keivanloo, Bipin Upahyaya

Department of Electrical and Computer Engineering, Queen's University, Kingston, Ontario, Canada
E-mail: {ying.zou, iman.keivanloo, bipin}@queensu.ca

Joanna Ng

CAS Research, IBM Canada Software Laboratory, Markham, Canada
E-mail: jwng@ca.ibm.com

Abstract: Nowadays, people use on-line services to conduct various tasks, such as on-line shopping and trip planning using web applications. Potentially, end-users compose ad-hoc business processes by integrating different web applications. Generally, end-users are required to enter information into web forms in order to interact with the web applications during the ad-hoc process composition. End-users often have to repetitively type same information into different web applications. It could be a tedious job for an end-user to fill in several web forms with identical information. To save end-users from typing redundant information and increase composition productivity, it is critical to propagate and pre-fill user inputs across different web applications. However, the existing approaches do not support this need well. Existing auto-filling approaches do not exploit all of the available resources useful to facilitate the auto-filling task. In this paper, we propose an intelligent auto-filling framework which propagates user's inputs across different web applications, collects end-user's inputs, identifies user's usage patterns, and detects end-user's contexts. Our framework clusters input user interface (UI) components to form semantically similar groups of input UI components. Through an empirical study, we show that our clustering approach for identifying similar input UI components achieves a precision of 89% and a recall of 83%. Furthermore, our framework achieves an average precision of 74.5% and an average recall of 58% on pre-filling web forms, and a precision of 82.25% and a recall of 68.4% on suggesting values to end-users if the end-users edit the initial pre-filled values.

Keywords: Web Form Auto-Filling; Clustering; Usage Patterns; Context-aware.

Reference to this paper should be made as follows: Shaohua Wang, Ying Zou, Iman Keivanloo, Bipin Upahyaya, and Joana Ng. (xxxx) 'An Intelligent Framework for Auto-filling Web Forms from Different Web Applications', *International Journal of Business Process Integration and Management*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Shaohua Wang is currently a PhD student in the School of Computing at Queen's University. His research interests include: Web Services, Web Mining and Mining Software Repositories.

Dr. Ying Zou is an associate professor in the Department of Electrical & Computer Engineering at Queen's University. She is Canada Research Chair in Software Evolution. She is a visiting scientist of IBM Centers for Advanced Studies. Her research interests include Software Engineering, Software Evolution & Maintenance, Software Empirical Engineering and Service Oriented Computing.

Dr. Iman Kanveiloo is a postdoc researcher in the Department of Electrical & Computer Engineering at Queen's University. His research interests include: Code Search, Software Clone and Mining Software Repository.

Dr. Bipin Upadhyaya is a postdoc researcher in the Department of Electrical & Computer Engineering at Queen's University. His research interests includes: Web Services.

Joanna Ng is the Head of Research at IBM Canada Software Laboratories, Center for Advanced Studies. She is a Senior Technical Staff Member of IBM Software Group. She is an IBM Master Inventor with a track record of profitable innovations.

1 Introduction

Recently, web services are playing an important role in people's life. Various activities, such as booking airline tickets and reserving hotel rooms, are conducted by end-users using web applications. Potentially, end-users conduct an ad-hoc business process by integrating a set of tasks to meet end-user's needs. Typically, such a set of tasks of an ad-hoc business process is not performed in a strict execution order. During the task integration of ad-hoc business processes, end-users generally interact with web services by entering information into them.

However, the end-users often have to repetitively type the same information into different services. A service often contains multiple input parameters receiving information provided by the end-users, such as the one illustrated in Figure 1. For example, planning a fishing trip is an ad-hoc process for many end-users. Planning such a trip could involve two tasks: renting a car and reserving a boat, and each task involves at least several input parameters. For example, an end-user has to input their personal information, such as the ones illustrated in Figure 1, into Enterprise [1] which is a car rental website to book a rental car. When the end-user reserves a boat on Gone Sailing Adventures [2] which is a boat renting website, he or she is also required to enter his or her personal information as illustrated in Figure 2. The requested personal information can be identical or similar in both websites. The end-user has to enter the same personal information again into the boat renting website although the end-user has entered it in the car rental website. It could be a tedious and repetitive job

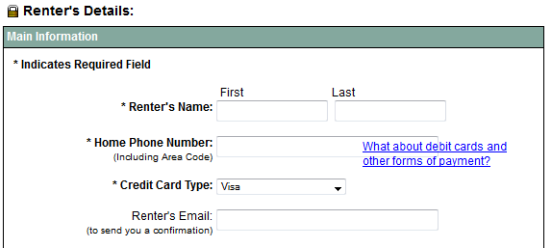

Figure 1: A sample screen shot of a web form requiring user's personal information to rent a car.

for an end-user to fill in a large number of web forms with multiple input UI components [3]. A large amount of information in the various services can be the same. To save end-users from such tedious process, it is more convenient for end-users if the information commonly required among different services can be propagated and pre-filled using the previously entered information across different web applications.

Recently, researchers in industry and academia have developed several tools and approaches to address the problem. Web browsers provide web form auto-filling tools, such as Mozilla Firefox Autofill Forms [4] and Google Chrome Autofill Forms [5], to help end-users fill

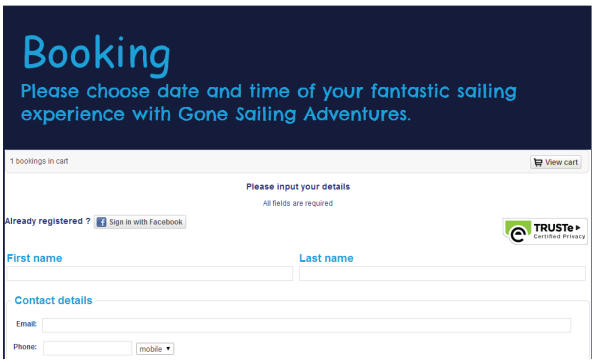

Figure 2: A sample screen shot of a web form requiring user's personal information to reserve a boat.

in web forms. In general, the auto-filling tools record the values entered by an end-user. The recorded information is used for pre-filling by identifying the potential reuse cases via string matching on parameters' names. The tools also allow users to modify the recorded information manually. Academic studies (*e.g.*, [6], [7], [8]) have also proposed approaches to help end-users fill in web forms. Hartmann et al. [6] present a novel mapping process for linking contextual information and user interface components of web forms. Toda et al. [7] present a probabilistic approach for automatically using data-rich text to fill form-based input interfaces. Winckler et al. [8] propose an approach to support the exchange of data between personal data and web forms. Specifically, auto-filling plays an important role on smart phones. Rukzio et al. [9] 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. However, all of these tools and approaches suffer from the following drawbacks:

- **Limited support of collecting and analyzing end-user's inputs automatically**: The existing approaches do not store and organize all of the available user inputs entered previously by end-users. Some tools such as Mozilla Firefox Autofill Forms [4] need end-users to manually create their personal data and preferences records. Typically, a record contains a key and its corresponding value, such as a key "City" and its value "Toronto". The tools fill the values of records into web services (*e.g.*, web applications) based on textual similarity between the keys of personal profile records and the input parameter of services. Indeed, some tools such as Google Chrome Autofill Forms [5] provide limited support on collecting user inputs and analyzing patterns of user inputs. However they are limited to basic personal information, such as credit card information.

- **Limited propagation of end-user's previous inputs to different web applications used by an end-user**: The existing approaches do not exploit all of the available information of

web services and user inputs for auto-filling. For example, a collection of web applications can be linked implicitly by end-users during the task integration of an ad-hoc process. An end-user usually purchases concert tickets from *ticket liquidator* [10] an event ticket booking website and then books bus tickets from *Greyhound* [11] a bus ticket booking website. These two web applications are linked implicitly by the end-user. If the end-user books two tickets from *ticket liquidator* and a few minutes later tries to buy bus tickets from *Greyhound*, the framework should be aware that the end-user would buy two bus tickets from *Greyhound* and pre-fill the number of required tickets (*i.e.*, two tickets) into *Greyhound* by learning from the data entered in the previous step of the implicit ad-hoc process composition. However existing approaches cannot support this type of intelligent filling.

In this paper, we propose a framework to address the aforementioned limitations. The proposed framework automatically detects user inputs and builds user profiles containing personal and preference information. Then our framework organizes and analyzes the user inputs to generate the patterns of user inputs for auto-filling during the task integration of ad-hoc processes. The framework explores the similarity between input user interface (UI) components among different web services. Understanding the relationships between input UI components is a crucial step for automatically propagating user inputs between different services. Furthermore, our proposed framework is context-aware and dynamically detects the changes of user's contexts, such as user's current location. Our framework supports the information exchange across user interface (UI) components (*e.g.*, drop-down box and text fields). In addition, the framework recommends a new list of values to end-users when the end-users modifies pre-filled values. To the best of our knowledge, we are the first to propose a comprehensive solution on service auto-filling during the task integration of ad-hoc processes.

This paper is an extended version of our earlier work [12]. which is a position paper published in IEEE 2013 International Workshop on Personalized Web Tasking on 2013 IEEE Ninth World Congress on Services (SERVICES). The original work:

- proposes the blueprint of our framework that analyzes user interfaces to extract information of input UI components, collecting user inputs and clustering input UI components to identify similar input UI components.

- conducts an empirical study to evaluate the effectiveness of the clustering algorithm on identifying similar user interface components which potentially require a same user input.

We extend the earlier work in the following aspects:

1) We redefined the design of our framework and provide the detailed description of our proposed framework.

2) We extended our framework that identifies patterns of user inputs, extracts user contexts and fills in web forms using previous user inputs based on user contexts and usage patterns.

3) We conducted an empirical study to evaluate the effectiveness of our framework for web form filling on a large and well-known dataset.

The rest of the paper is organized as follows. Section 2 introduces the structure of web forms. Section 3 presents an overview of our proposed framework. Section 4 introduces the empirical study. 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 Forms

Nowadays, end-users with no IT background compose ad-hoc processes by integrating different tasks of web applications to meet their needs. In this paper, we propose a framework to help end-users fill in values to web forms during the task integration of ad-hoc process composition.

Typically, an end-user interacts with web applications by entering data to them. Usually, the web user interface of a web application is written in HTML (or XHTML) and Cascading Style Sheets (CSS) and contains web forms. A web form consists of a set of input UI components, such as text fields, radio buttons and check-boxes. A user interface can be converted into HTML DOM tree [13]. The developers use an HTML tag <form> to define a web form. An input UI component is expressed as an <input> element in an HTML DOM tree. The <input> element has several attributes, such as *name* specifying the name of an <input> element, *type* defining the type of the <input> element (*e.g.*, a button or checkbox), and *value* stating the value of an <input> element. Furthermore, an <input> element is associated with a human-readable label, such as "Renter's Name" illustrated in Figure 1. We can obtain all the information of a web form and its associated input UI components by parsing the HTML source code.

3 Our Proposed Framework

This section presents our proposed framework. First, we introduce the major components of our framework. Second, we present the interactions among the components. Figure 3 shows the architecture of our framework and interactions among its components.

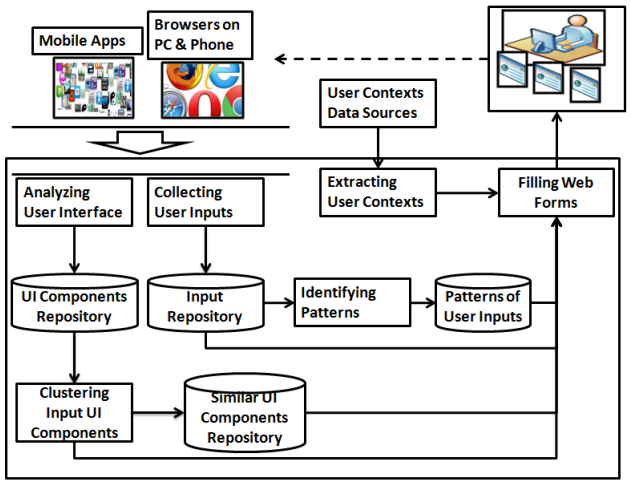

Figure 3: Overview of our proposed framework for filling web forms

3.1 Major Components of Our Framework

Our framework consists of six major components listed as follows:

- ***Analyzing User Interfaces.*** This component parses the web pages visited by an end-user and then extracts information (*e.g.*, a label) of the input user interface (UI) components expecting an input from an end-user. All of the collected entities are stored in *UI Components Repository*.
- ***Collecting User Inputs.*** This component detects and collects the inputs from end-users, stores the user inputs for further analysis. All of the collected user inputs are stored in *Input Repository*.
- ***Identifying Patterns of User Inputs.*** The framework analyzes the collected user inputs in *Input Repository* and identifies patterns of user inputs.
- ***Identifying Similar Input UI Components.*** The collected input user interface components in *UI Components Repository* are clustered into semantic groups. The input UI components within a group are similar to each other and potentially require the same value. The semantic groups are stored in *Similar UI Components Repository*.
- ***Extracting User Contexts.*** The framework detects and extracts user contexts from external sources such as a user's calender.
- ***Filling in Web Form.*** The framework conducts web form filling using the collected user inputs, patterns of user inputs, clusters of similar input UI components, and user contexts such as entries in Google Calendar.

3.2 Interactions Among Components

In this sub-section, we introduce the interactions among components through the scenarios of end-users' interactions with input UI components of web forms. After an end-user opens a web page, we summarize three scenarios:

Scenario 1. Before the end-user enters a value into a web form: The framework conducts the following analysis:

First, the framework starts ***analyzing the user interface*** to extract the textual information of input UI components.

Second, the extracted information of an input UI component is used to check whether the input UI component is already stored in *UI Components Repository* or not.

- If it is not in *UI Components Repository*, the framework conducts the following steps:
  - Step 1. The input UI component with its extracted information is stored in *UI Components Repository*.
  - Step 2. The framework starts *identifying similar input UI components* by clustering the input UI components in *UI Components Repository* and stores the clusters of similar input components in *Similar UI Components Repository*.
  - Step 3. The framework searches for the similar input UI components, associated with some previous user inputs in *Input Repository*, from *Similar UI Components Repository*. If such input UI components are identified, the framework saves them as a *Candidate UI Set*. Otherwise the input UI component cannot be pre-filled with any previous inputs.
- If it is already stored in *UI Components Repository*, the framework checks whether the input UI component is associated with any previous user inputs in *Input Repository* or not.
  - If the input UI component is associated with a set of previous user inputs which can be entered by the end-user at different times, the framework saves the set of user inputs as a *Candidate Set*.
  - If the input UI component is not associated with any previous user inputs in *Input Repository*, the framework repeats **Step 3** to identify the similar input UI components.

Third, the framework starts ***extracting user contexts*** from different data sources such as Google Calendar.

Fourth, the framework starts ***filling in web forms*** using the extracted user contexts and the *Candidate Value* or *Candidate UI Set*.

Scenario 2. When the end-user enters a value into a web form: When the end-user enters a set of inputs to a web form of the web page and submits the web form, the framework starts ***collecting the user inputs***. The framework stores the set of inputs and the information of UI components receiving the user inputs in the *Input Repository*. Then the framework starts ***identifying patterns of user inputs*** by analyzing the collected user inputs in the *Input Repository*.

Scenario 3: When the end-user is not satisfied with a pre-filled value: The end-user modifies the pre-filled value. During the modification of the initially pre-filled value, the ***filling in web forms*** component dynamically recommends a list of possible user inputs which are potentially suitable for the input UI component.

The remainder of this section elaborates on each component of our framework.

3.3 Analyzing User Interfaces and Collecting User Inputs

Figure 4: An example of a HTML page

Due to the high diversity in the structure of HTML web pages, locating a label for an input user interface component is a challenging and difficult task. A label of a user interface component can be positioned in different locations. Some websites explicitly link a label and its corresponding user interface component using the HTML tag <label>. A label can be assigned to an element either by using the "for" attribute, or by placing the element inside the <label> element. In this case, it is easy to identify the label for the user interface component. However, we found that most of websites do not link the label and its corresponding user interface component explicitly. We have to use heuristics to identify the label representing an input element. We adopt the approach used in [14].

The approach in [14] analyzes different parts of a web page. A web page consists of a set of opening and closing HTML tags and these tags separate the web page content into different partitions. The approach traverses and analyzes the HTML DOM tree [13] to identify the partitions with a label. When reaching a partitioning element, such as $< p >$, $< br >$, and $< hr >$, the approach creates a label and adds the text node inside the partitioning element to the label. If the partitioning element has a child partitioning element, the approach links the text nodes appearing under the child partitioning with the child partitioning element. For each input element, the approach calculates the distance between the input element and the text nodes to determine which text node is for the input element. The distance is calculated based on the number of nodes visited to reach the input element from a text node. The approach chooses the label with the least distance as the label for the input element.

3.3.1 Analyzing User Interfaces

Our framework extracts input user interface (UI) components of web pages visited by an end-user. The framework parses the HTML DOM [13] of a web page to extract input user interface (UI) components. For an input UI component such as component 1 in Figure 4, the following information is extracted:

- *Textual information of a web form.* We extract the information of a web form where the input UI component locates as follows:
  - The value of the name attribute of HTML tag $< form >$. In XHTML, we extract the id attribute instead.
  - The label or text (a human readable textual information) assigned for the web form. For example, the label "Sign in" illustrated in Figure 4 is a label for the sign-in form of ebay [15].
  - The URL of the web page where the web form is in.
- *Information of the Input UI Component* We store the following information:
  - *Type.* We keep the type of the UI component such as drop-down box and text field.
  - *Coding Information.* We store the values of the attributes of the input element of HTML DOM such as id, name, text and hint. We locate the HTML INPUT tag <input> specifying a user interface component (*i.e.*, the component receiving input from end-users) to extract the coding information.
  - *Label.* We extract the label associated with the user interface component.

The extracted information is stored in *UI Components Repository* which holds a collection of input UI components. We define a unique id for every input

UI component and web form. A unique id of an input UI component is used to locate it in the *UI Components Repository*. The unique id of a web form is used to identify the neighbor input UI components for an input UI component. If two input UI components reside in the same web form, these two input UI components are neighbors to each other. We generate a unique id for every input UI component and every web form. We combine the values of an input component: *Label*, *Coding Information*, *Type* to be an id. We use the Textual information of a web form as its id. In the *UI Components Repository*, we also store the *Unique IDs of neighbors* for a input UI component.

3.3.2 Collecting User Inputs

To make a framework intelligent, it is important to track and collect user inputs from web forms. When an end-user enters a user input to an input UI component, we extract the following information:

- *User Input.* We store the value of the user input entered by an end-user.
- *Information of Input User Interface Component.* We store the exact same types of information of an input user interface component as the ones stored in Section 3.3.1 (*i.e.*, analyzing user interfaces).
- *Time Stamp.* After the end-user enters a user input and submits the user input through a web form, we store the time of the submission. Usually a set of user inputs entered into a web form has the same time stamp because these user inputs are submitted together by end-users.

We use the same approach described in Section 3.3.1 (*i.e.*, analyzing user interfaces) to generate a unique id for an input user interface component. We use the generated id to locate the input component in *UI Component Repository*, then we store the user input, the id of its associated input UI component, and the *Time Stamp* in *Input Repository*.

We modify and extend an open source tool called Sahi [16] used for automating web application testing to collect end-user's information and analyze user interfaces. Sahi injects Javascripts into web pages using a proxy and the Javascripts help automate the actions of web applications. It monitors the user's actions (*e.g.*, click, submit and search), remembers the possible user's inputs (*e.g.*, search queries, end-user's name, age, gender), and generates a log. By default, the Sahi tool does not include the detailed information (*e.g.*, descriptive information) of an input user interface component in the log, we extend the tool to extract more information of the input user interface components.

3.4 Identifying Patterns of User Inputs

When an end-user conducts web tasks such as on-line shopping for a while, a lot of user inputs can be collected.

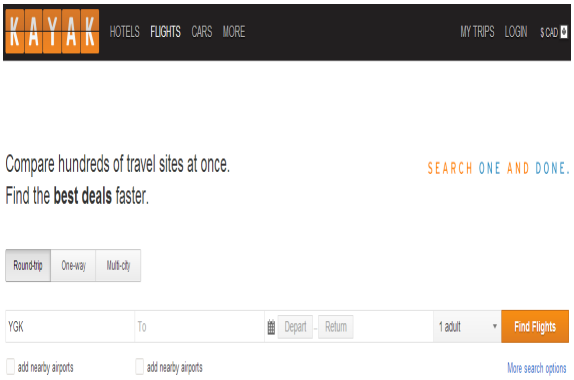

Figure 5: A sample screenshot of a web form

Among these user inputs, there exist possible patterns of user inputs. The user inputs within a pattern can be filled together into a web form if any user input is selected by the end-user. For example, an end-user searches for cheap flights using the search form in Figure 5 of KAYAK [17] a travel website. The end-user always searches for a round-trip business class flight ticket from Toronto to Los Angeles and a round-trip economic class flight ticket from Toronto to Vancouver.

In this study, we mainly focus on recognizing the patterns of user inputs from web forms. To identify the patterns of user inputs from the *Input Repository* and *UI Component Repository*, we conduct the following steps:

- *Step 1.* Every user input in the *Input Repository* has a time stamp. We use the time to group user inputs. A group of user inputs are submitted at the same time.
- *Step 2.* Every input UI component in the *UI Component Repository* has an id of a web form. We use the web form id to group input UI components. A group of input UI components are from the same web form.
- *Step 3.* We use the unique ids of input UI components to identify the user inputs associated with the groups of input UI components generated in Step 2. The output of this step is the mapping between a web form (*i.e.*, a set of input UI components) and groups of user inputs entered to the web form at different times.
- *Step 4.* We identify the frequent patterns of user inputs of a web from using the BI-Directional Extension based frequent closed sequence mining (BIDE) pattern mining algorithm proposed in [18]. The BIDE mines maximal frequent patterns. For example, a web form in Figure 5 takes user inputs to search for cheap flights. Over the time, for example, there could be three sets of user inputs entered into the web form. The three sets of user inputs are:
  Set 1: "round-trip, LA, NYC, Thr 13/3, Tue 22/4, 1 adult".

Set 2: "round-trip, Toronto, Paris, Sat 26/4, Fri 27/6, 2 adults".
Set 3: "round-trip, Toronto, Paris, Tue 9/12, Wed 31/12, 2 adults".
We set BIDE to identify any pattern of user inputs which shows up more than twice. Therefore the pattern generated for the above example is "round-trip, Toronto, Paris, 2 adults".

3.5 Identifying Similar Input UI Components

The framework clusters the input UI components stored in the *UI Components Repository* to form semantic clusters using their textual information. The input UI components within a cluster are semantically similar and potentially require a same value. Each input UI component has the *Coding Information* and *Label*. We merge all of the descriptive textual information together from *Coding Information* and *Label* to construct *a bag of words* [19] for each input component.

Our clustering approach weights each word. A bag of words of an input UI component can be represented as a vector of weight values. We use the Cosine Similarity [20] to calculate the similarity between two vectors of input UI components and Quality Threshold Clustering Algorithm [21] to cluster similar UI Components.

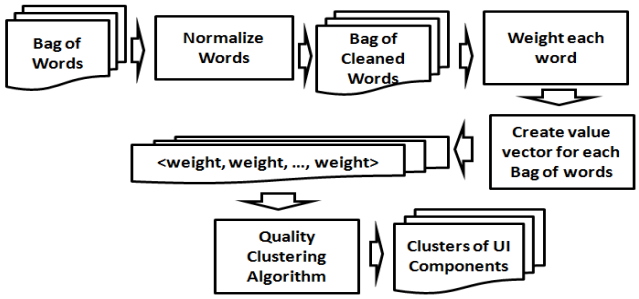

Figure 6: Overview of our approach for clustering UI components

Figure 6 shows the major steps of our clustering approach consisting of the following steps:

1) Words Normalization: We split the words in the bags of words stored in the *UI Components Repository* by special characters such as "_" and capital characters if applicable. We use Wordnet [22] an lexical database for English to remove non-English words. The stop words removal, word stemming are conducted in this step to normalize the words of each component. A bag of "cleaned" words is generated for each component.

2) Value Vector Creation: The Term Frequency and Inverse Document Frequency (TF-IDF) value are used to calculate the weight for each word in the bag of "cleaned" words. The *Term Frequency* (TF) is the frequency of a word appearing in a document. We refer to a bag of words as a document. The *Inverse Document Frequency* (IDF) diminishes the weight of words that occur very frequently in the whole corpus and increases the weight of words occur rarely. We calculate the TF as shown in Equation (1) and the IDF as shown in Equation (2) for each word.

$$TF = \frac{|\{occurrences\ of\ a\ word\ in\ the\ document\}|}{|\{total\ words\ in\ the\ document\}|} \quad (1)$$

$$IDF = \log \frac{|\{documents\ in\ the\ corpus\}|}{|\{documents\ having\ the\ word\}|} \quad (2)$$

We use a TF-IDF value as the weight of a word. We calculate the weight for each word as shown in Equation (3).

$$weight = TF \times IDF \quad (3)$$

Once every word has a weight, a bag of words is expressed as a vector of values, and an input UI component is represented by a vector of values.

3) Quality Threshold Clustering [21]: We use the Quality Threshold (QT) Clustering algorithm to cluster the similar UI components. We decide to use QT because it returns the consistent results across multiple runs of clustering with the same input, and it can be used to cluster particular groups. However, requiring more computation resources is a drawback of QT. We measure the similarity $Sim(C_i, C_j)$ between UI components C_i and C_j using the cosine similarity algorithm [20]. The $Sim(C_i, C_j) = Cosine(V_i, V_j)$, where V_i is the value vector of UI $component_i$ and V_j is the value vector of UI $component_j$. We calculate the similarity value between any pairs of UI components except the components from the same parent element in HTML DOM Tree. The QT algorithm clusters UI components based on the similarity values of UI components. Within a cluster, all of the components potentially require the same inputs from users.

3.6 Extracting User Contexts

To make it context-aware, the framework extracts end-user's contexts and uses them for web form filling. The contextual information ranges from physical data such as user's location to "virtual" data such as user's to-do list. We identify three types of data resources which can be used for user's contexts extraction under the context of web form filling. The three sources are listed as follows:

- *The End-user's Computing Environment.* The end-users can use different computing environments for different web form filling tasks. The data extracted from the computing environments can help web form filling techniques identify the different web form filling scenarios of an end-user. For example, an end-user searches for the cheapest tickets using his or her computer at work and pays for the tickets using his or her own computer because his or her work computer is monitored by the corporate intra-net and payment information is too private. Furthermore, the configuration data, such as time zone and IP address can be used for filing input UI components requiring the current contexts (*e.g.*, current time, current location) from the end-user.

- *End-user's Applications.* End-users use various applications (*e.g.*, Google calender and Facebook) for their daily tasks such as sending emails and checking friend's references on Facebook. The information from such applications helps the web form filling techniques infer the details of end-users. For example, an end-user has blocked a time window ranging from Mar 09th to Mar 14th on his or her calender for a business travel. He or she wants to book an appointment with an optometrist through the optometrist's website. However only the time window from 3:00PM to 3:45PM on Mar 10th is shown as open on the website within the whole month of March. Therefore the time window 3:00PM to 3:45PM on Mar 10th should not be selected automatically.

- Web browsers: End-users use web browsers to surf web pages. The web browsers cache the users' bookmarks and historical activities. These information reflects the user's preferences. The web form filling techniques consult with the user's preferences during the process of filling web forms.

One challenge is to map the contextual data to the input UI components. In most cases, the descriptive text used in user interface differs from the one used to describe the context [6]. For example, the location information in a calender entry has the label "Location" whereas the location information for the UI in a car rental website is "pick-up". We adopt the approach proposed in [6] for mapping contextual information to UI elements. We combine string-based and semantic similarity measures.

In our current work, we only extract simple user contexts: current location, current time and calendar entries in Google calendar.

3.7 Filling in Web Forms

After collecting and analyzing user inputs, user's contexts and input UI components, the framework starts filling in the web forms. During the process of filling values to web forms, there are two strategies listed as follows:

Strategy 1: Pre-filling a web form before an end-user enters a value. When the framework receives current user contexts and the possible *Candidate Set* or *Candidate UI Set* as described in Section 3.2. The following steps are conducted to fill an input UI component, denoted as ***target component***, of a web form with a value: First, the framework detects whether the ***target component*** requires the current user contexts (*e.g.*, current location and current time) or not. If the ***target component*** requires the current user contexts, then the framework pre-fills a current user context to the ***target component***. Otherwise, the framework processes a *Candidate Set* or *Candidate UI Set* in the following way:

- The *Candidate Set* of the ***target component*** is parsed if the ***target component*** is already stored in the *UI Component Repository*. The framework ranks the user inputs of *Candidate Set* based on their timestamps from latest to earliest and pre-fills the latest user input to the ***target component*** if the latest user input does not contradict with any calender entries.

- The *Candidate UI Set* of the ***target component*** is parsed if the ***target component*** is a new input UI component in the *UI Component Repository*. The framework chooses the latest user input of the most similar input UI component to the ***target component***.

Strategy 2: Recommending values to input UI components after an end-user adjusts any pre-filled values. The following steps are conducted to fill an input UI component, denoted as ***target component***, of a web form with a value:

- Step 1. The framework builds a set of user inputs for recommendation to the ***target component*** in the following steps:
  - If a *Candidate Set* exists, the *Candidate Set* is considered as a recommendation set.
  - If a *Candidate UI Set* exists, all of the user inputs of the input UI components in *Candidate UI Set* are extracted and built as a recommendation set.
  - Otherwise there is no set of user inputs for recommendation.

- Step 2. The framework recommends a user input to the ***target component*** based on the modified value (*i.e.*, a string) in the ***target component*** dynamically. The framework conducts a string matching between the modified value and the values in the *recommendation set* to identify the suitable values from the *recommendation set*.
  - If no values identified, no value is recommended to the ***target component***.
  - If a set of values identified, the values are ranked based on their times from latest to earliest.

- Step 3. If the end-user selects a user input from the recommendation set, the framework searches for the patterns of user inputs containing the selected user input and fills the other values in the pattern to other input UI components which are neighbors to the ***target component***. If multiple patterns identified, the most frequent pattern is chosen.

4 Experiment

In this section, we conduct several experiments to evaluate the effectiveness of our framework. First, we

introduce the experiment setup. Then, we present the research questions. For each research question, we state the motivation of each question, the analysis approach and the corresponding results.

Table 1 Descriptive statistics of TEL-8 web forms.

Domain	# of Forms	# of Components
Airfare	65	447
Auto Mobiles	47	706
Books	84	441
Car Rentals	25	204
Hotels	39	396
Jobs	49	389
Movies	73	504
Music Records	65	448

4.1 Experiment Setup

In this sub-section, we discusses our data collection and processing.

4.1.1 Web Forms Dataset

We conduct our study on TEL-8 Query Interfaces [23] which contain a set of web query interfaces of 447 web sources across 8 domains in the web. The 8 domains are Airfares, Automobiles, Books, Car Rentals, Hotels, Jobs, Movies, and Music Records. We chose this dataset as our testing data due to the two following reasons:

- The dataset contains web interfaces (*i.e.*, web pages) from the typical domains of web applications that capture web form strucures on the web nowadays. Moreover, the dataset is publicly available.
- The dataset has been widely used by several research studies in different research areas. For example, Zhang et al. [24] extract the query capability from this dataset for understanding the web query interfaces; He et al. [25] propose a correlation mining approach to discover complex matchings across web query interfaces using this dataset; Araujo et al. [26] propose a concept-based approach for web form filling.

The dataset of TEL-8 Query Interfaces is manually collected in [24]. We extract input UI components from the web sources. Table 1 shows the number of web forms of each domain and the number of extracted input UI components from the web forms of each domain. In total, we collected 3,535 input UI components.

4.1.2 User Input Collection and Processing

To test the effectiveness of our framework on web forms auto-filling, we collect user inputs from four end-users. The authors of this paper are not involved in the user input collection. The four end-users are all graduate students, who typically spend 8-10 hours per day on-line.

The four end-users randomly selected web forms from TEL-8 dataset and used the approach of ***collecting user inputs*** described in Section 3.3.2. The four end-users spent two days on collecting user inputs. We collected four end-user profiles for each domain. Based on the four profiles for each domain, we generate more user inputs for each profile by randomly splitting the collected user inputs and then combining the split user inputs.

4.2 Research Questions

In this sub-section, we presents our two research questions. For each research question, we discuss the motivation of the research question, analysis approach and the corresponding findings.

RQ1. Is our clustering technique effective in grouping similar UI components?

Motivation. Similar user interface components may consume the same values. Our proposed framework clusters input UI components into semantic groups. The components within a cluster are considered to be similar. In this research question, we verify the effectiveness of the clustering approach on identifying similar components.

Analysis Approach. To test effectiveness of our clustering approach, we apply our clustering approach on the extracted input UI components of each domain. To be able to validate the results manually, we randomly sample input UI components with confidence level 95% [27]. In total, we randomly sampled 347 input UI components from 3,535 input UI components. Second, we perform a manual clustering step on the sampled 347 components to create our *gold standard* for validation purposes. Third, we apply our clustering approach on the sampled 347 components to automatically cluster them into different semantic groups. During the clustering process, we use 0.85 determined experimentally as the minimum similarity threshold between any pair of UI components in a cluster. Forth, we compare the results generated from the last step with the *golden standard*. We compute precision and recall of our clustering approach using Equation (4) and Equation (6) respectively.

$$Precision = \frac{\sum_{i \in C} P_{C_i}}{Length(C)} \tag{4}$$

$$P_{C_i} = \frac{succ(C_i)}{succ(C_i) + mispl(C_i)} \tag{5}$$

$$Recall = \frac{\sum_{i \in C} R_{C_i}}{Length(C)} \tag{6}$$

$$R_{C_i} = \frac{succ(C_i)}{succ(C_i) + missed(C_i)} \tag{7}$$

where C_i is the cluster i, P_{C_i} and R_{C_i} are the precision and the recall for cluster C_i. $succ(C_i)$ is the number

of input fields successfully placed in the proper cluster C_i. $mispl(C_i)$ is the number of input fields incorrectly clustered into C_i. $missed(C_i)$ is the number of input fields that should be clustered into C_i, but incorrectly placed into other clusters. Length(C) is the number of clusters.

Results. The clustering approach generates 107 clusters in total. All the components within a cluster potentially require the same user input from the end-users. If one of them has been associated with a previous user input, it is most likely that the other components of the cluster require the same user input. Our approach achieves a precision of 89% and a recall of 83%. Based on a postmortem manual investigation, we found that the major reason for missing some cases is that some of the extracted UI components, in our test data, do not have a meaningful descriptive information. This is the major reason that the values of precision and recall are not 100%.

RQ2. Is our framework effective in pre-filling web forms?

Motivation. Pre-filling web forms improves the productivity of composing ad-hoc processes by saving end-users from repetitive inputing of values to web forms [28]. In this research question, we test the effectiveness of our framework on pre-filling web forms based on the previous user inputs, patterns of user inputs, user's contexts and clusters of similar input UI components.

Table 2 Results of our framework and Firefox Autofill Forms on pre-filling web forms. P stands for Precision and R stands for Recall.

Domain	Our Framework		Firefox	
	P(%)	R(%)	P(%)	R(%)
Airfare	62	55	45	30
Auto Mobiles	70	58	55	38
Books	78	64	69	55
Car Rentals	82	65	75	45
Hotels	79	53	60	45
Jobs	81	63	79	48
Movies	71	55	60	22
Music Records	73	54	48	26

Analysis Approach. To test the effectiveness of our framework on pre-filling web forms, we compare our framework with Mozilla Firefox Autofill Forms [4] used as a baseline approach. We conduct the experiment in the following steps. First, we apply our approach of ***identifying patterns of user inputs*** on our collected user inputs in Section 4.1.2 (*i.e.*, User inputs collection and processing). Second, we correct the incorrectly placed input UI components from the clusters generated in *RQ1* and use the clusters for identifying similar input UI components in our framework. Third, we manually enter values to the predefined entries of the default profile in Mozilla Firefox Autofill Forms [4]. Fourth, we apply our framework to pre-fill the web forms which are not selected by the four end-users during the collection of user inputs in Section 4.1.2 using the collected user inputs in Section 4.1.2. Fifth, we manually use Firefox Browser to visit the web forms used in the previous step, and Firefox Autofill Forms to fill in input UI components of the visited web forms. Last, we calculate the precision and the recall using Equation (8) and Equation (9) to measure the effectiveness of our framework and Firefox Autofill Forms on web form pre-filling.

$$Precision = \frac{|Correct\ Filled\ Input\ Components|}{|Filled\ Input\ Components|} \quad (8)$$

$$Recall = \frac{|Correct\ Filled\ Input\ Components|}{|Input\ Components\ Need\ To\ Be\ Filled|} \quad (9)$$

Results. Table 2 shows that our framework outperforms Firefox Autofill Forms on pre-filling web forms. On average, our approach can achieve a precision of 74.5% and a recall of 58%, the Firefox Autofill Forms can achieve a precision of 61.3% and a recall of 38.6%. We investigated our results and found that even simple end-user context information such as current location helps identify right values for input UI components.

The main reason of the low recall of Firefox Autofill Forms is lack of support for complex types of information. The Firefox Autofill Forms is limited to basic types of information, such as Name and City, for web form filling. Furthermore, the Firefox Autofill Forms can only keep one value for each type. However an input UI component may require different values of a specific type under different tasks. For example "contact phone number" can have two values, one value is used as a travel contact number and the other one is used as the contact number of receiving shipments from E-commerce websites. Therefore, the Firefox Autofill Forms does not distinguish the different pieces of information of one type under different tasks. In addition, the Firefox Autofill Forms solely relies on keyword string matching when identifying user inputs for input UI components. If the textual description of UI components is missing or does not contain the keywords predefined in the default user profile of Firefox Autofill Forms, the Firefox AutoFill Forms missed the opportunity for pre-filling.

RQ3. Is our framework effective in recommending values to web forms?

Motivation. When an end-user is not satisfied with a pre-filled value to an input UI component, he or she would modify the pre-filled value. During the course of value modification, the framework recommends a list of values to the end-user. In **RQ2**, we evaluate the effectiveness of our approach on web form auto-filling. In this research question, we test the effectiveness for recommending values to end-users.

Analysis Approach. To test the effectiveness of our framework on recommending values to end-users, we compare our framework with Google Chrome Autofill Forms [5] used as a baseline approach. We conduct the experiment in the following steps. First, we clear out all of the input UI components which are incorrectly

pre-filled in **RQ2**. Second, we build a new set of user inputs for recommending values to UI components by excluding the user inputs used in **RQ2**. Third, we apply our approach of ***filling web forms*** on recommending values to the UI components incorrectly pre-filled in **RQ2**. Fourth, we manually visit the web forms in Section 4.1.1 which do not contain a UI component incorrectly pre-filled in **RQ2** using Google Chrome Web Browser and enter the new set of user inputs into proper UI components. Fifth, we apply Google Chrome Autofill Forms on the input UI components incorrectly pre-filled in **RQ2**. Finally, we calculate precision and recall using the Equation (8) and Equation (9) based on the value recommended on the top (*i.e.*, We choose the top value for the second time pre-fill), to measure the effectiveness of our framework and Google Chrome Autofill Forms on suggesting values to end-users.

Table 3 Results of our framework and Google Chrome Autofill Forms on recommending values to end-users. P stands for Precision and R stands for Recall.

Domain	Our Framework		Chrome	
	P(%)	R(%)	P(%)	R(%)
Airfare	73	68	65	55
Auto Mobiles	82	63	75	60
Books	86	72	80	65
Car Rentals	87	75	85	70
Hotels	82	65	80	60
Jobs	90	81	70	55
Movies	78	60	66	48
Music Records	80	63	62	52

Results. Table 3 shows that our framework outperforms Google Chrome Autofill Forms. When the pre-filled values are not correct in the first place (*i.e.*, filling web forms in Scenario 2), on average, our approach can achieve a precision of 82.25% and a recall of 68.4%, and Google Chrome Autofill Forms can achieve a precision of 60.3% and a recall of 58.1%, on recommending values to end-users. We further investigate the results of our framework and Google Chrome Autofill Forms. We found that the patterns of user inputs help identify the right values for recommendation. With the accumulation of user history and more patterns generated, the results can be improved gradually. The Google Chrome Autofill Forms can only track a limited number of types of user inputs, such as name and address. The limited support of user input collection leads to the low recalls.

RQ4. Does our framework have qualitative advantages over existing web browser auto-filling tools?

Motivation. In **RQ2** and **RQ3**, we compare our framework with Mozilla Firefox Autofill Forms [4] and Google Chrome Autofill Forms [5] quantitatively. In this research question, we compare our proposed framework with Mozilla Firefox Autofill Forms [4] and Google Chrome Autofill Forms [5] qualitatively.

Analysis Approach. To answer this research question, we follow the analysis framework proposed in [29] to compare our framework with the mentioned two tools. Although the analysis framework in [29] is proposed for comparing web information extraction (IE) tools, we extend and apply it on the comparison of web auto-filling techniques.

We compare tools in three dimensions: *Task Difficulties*, *The Techniques Used* and *Automation Degree*. The first dimension, *Task Difficulties*, evaluates the difficulty of an auto-filling task, which can be used to answer the question "why an auto-filling technique fails to handle some input UI components?" The second dimension, *The Techniques Used*, compares the techniques used in different auto-filling tools. The third dimension, *Automation Degree*, evaluates the effort made by end-users for providing values to input UI components. For each dimension, we include a set of features that can be used as criteria for comparing and evaluating auto-filling techniques.

We list the features of each dimension as follows: **Task Difficulties.** The range of types of user interface components which can be identified and filled in with a value is important to an auto-filling tools. In this paper, we classify the range into three lengths based on the number of user interface components: short (1 to 5 UI components), medium (5 to 15 UI components), long (over 15 UI components). Supporting more UI components require tremendous research effort. Due to the variety of HTML page structures, the task of extracting meaningful textual information for different UI components is not trivial. It is also a challenging job to match values with proper components, convert the values into right format for filling.

The types of data which could be used for filling in web forms can be previous user inputs, basic personal information profile (*e.g.*, name, address, credit card information), end-users' contextual data, user usage data. Each type of data source requires auto-filling tools to make different research efforts such as approaches of collecting data. The user usage data can also be viewed as a type of end-users' contextual data, however it requires relatively huge computation and history, we follow the approach in [30] to separate the user usage data from end-user's contextual data.

The Techniques Used. Filling web forms involves a series of algorithms. In summary, the features for comparing auto-filling tools from the perspective of techniques used include: tokenization, extraction rules, features involved, and learning algorithms. Furthermore, pre-filling and recommending are the two main strategies of reusing use data for filling values to web forms [26].

Automation Degree. Speaking of the task of filling values to web forms, it is really all about how much convenience the auto-filling tools can bring to end-users in terms of reusing their data. We identify three automation degrees on the data types supported by the auto-filling tools. The three automation degrees are listed as follows:

Table 4 Comparison Our Framework with Existing Tools: Firefox Autofill Form and Google Chrome Autofill Form

Tools	Automation Degree	Range of UI Components	Data Supported
Firefox	Partial	Short	Previous User Inputs Basic Personal Information
Chrome	Fully	short	Partial Previous User Inputs Partial Basic Personal Information User Usage Data
Ours	Fully	medium	Previous User Inputs Basic Personal Information Basic Contextual Data User Usage Data (maximum length of patterns)
Tools	Matching Algorithm	Learning Algorithm	Filling Strategy
Firefox	String Matching	String alignment	Pre-filling
Chrome	String Matching	String alignment Pattern mining	Recommending
Ours	String Matching Semantic Similarity	String alignment Pattern mining Clustering	Pre-filling Recommending

- *Fully Automation.* There is no need of human intervene on data management, such as user input collection.
- *Partial Automation.* The data management still needs some human intervene.
- *No Automation or very little.* End-users have to manually manipulate the data, such as creating personal profiles manually for filling web forms.

Results. Table 4 shows that our framework supports more types of data and user interface components than the other tools do. The Google chrome auto-filling tool only supports very limited number of user inputs (*e.g.*, credit card information and login credentials), and tracks all of the patterns of the user inputs supported by chrome (*i.e.*, Chrome keeps all lengths of patterns of user inputs). We believe the big advantage of our tool over the other existing two tools is analyzing and linking similar user interface components which enables the propagation of user inputs across different tasks during the ad-hoc business process composition.

5 Threats to Validity

This section discusses the threats to validity of our study following the guidelines for case study research [31].

Construct validity threats concern the relation between theory and observation. In this work, the construct validity threats are mainly from Parsing HTML web forms: Due to the various structures of HTML web pages, it is a challenging task to extract information from web pages. For example, the positions of labels in web form depend on the designer of the web page. The labels can be placed above, below, to the left, or to the right of an input element. We adopt the approach in [14] to extract information from web pages. The *User Input Collector* and *User Interface Analyzer* deal with the HTML web pages.

Threats to internal validity concern our selection of subject systems, tools, and analysis method. First, we modified the web application testing tool called Sahi [16] to trace the user inputs and extract the information we need for further process. We chose the Sahi instead of developing a new tool due to two reasons:

- We only have limited resources (*e.g.*, man power).
- The tool covers all types of devices, which becomes an advantage if we collect user inputs from a mass of end-users. In the near future, we plan to recruit more people for our user case study.

Second, we collected user inputs through four end-users. The number of subjects may be low, the main reason is that people are reluctant to give away their personal information. However, the number of end-users profiles is not low. We collected 32 real end-user profiles in total. Moreover, we generated more user profiles by splitting and combining the collected real personal profiles. Third, we performed a manual clustering step to create gold standard for validation, two human experts are involved in the creation and verification of gold standard. The human judgment could threat the validity, however it is very common to include manual study in research studies such as [26].

Reliability validity threats concern the possibility of replicating this study. We attempt to provide all the necessary details to replicate our study. The dataset of web forms used in our study are publicly available [23].

6 Related Work

In this section, we summarize the related work on form auto-filling tools and approaches.

Some tools from industry aim to ease user's pain in form filling. RoboForm [32] is specialized in password management and provides form auto-filling function. LastPass [33] is an on-line password manager and form filler. 1Password [34] is a password manager integrating directly into web browsers to automatically log the user into websites and fill in forms. These three tools store user's information in central space, and automatically fills in the fields with the saved credentials once the user revisit the page. However all of the tools above need users to create personal profiles manually and are not context-aware. Our framework can detect and analyze user's inputs automatically to generate user's profiles..

Some studies (*e.g.*, [8], [3], [35]) explore the use of semantic web technology for developing data binding schemas. The data binding schemas are essential techniques helping connect user interface elements with data objects of applications. The main drawback of this technology is that an ontology is needed before performing the data integration. The creation of ontology is time consuming. Instead of focusing on custom ontology for particular web applications, some binding schemas rely on the emergence of open standard data types, such as Microforms [36] and Microdata [37]. Winckler et al. [8] explore the effectiveness of the data schemas and the interaction techniques supporting the data exchange between personal information and web forms. However these approaches require some manual work on creating ontologies or annotate web forms using the open standard data types. Our framework does not require any manual work.

Some studies such as [38] require Apriori algorithm [39] tagging of websites, or a manually crafted list that includes the labels or names of input element to describe a semantic concept. These approaches can only be applicable to a specific domain or need explicit advice from the user. Hartmann et al. [6] present a novel mapping process for matching contextual data with UI element. Their method can deal with dynamic contextual information like calendar entries. We adopt M. Hartmann's mapping process for user's contextual information and UI elements.

Some studies such as [40] and [41] propose different taxonomies of context-aware data which gives guidelines to developers and researchers. In this study, we adopt the taxonomy proposed in [40] and identify the proper contextual data suitable for web form filling.

7 Conclusion and Future Work

In this paper, we propose an intelligent framework to help users fill in web forms and save them from repeatedly entering the same information to web forms in web applications. Our framework automatically collects user inputs and analyzes them for generalizing patterns. The patterns of user inputs and user context data are used to support web form auto-filling. Our framework reuses the previous user inputs by clustering the similar UI components into semantic clusters. Based on the results of our empirical study, our clustering algorithm can achieve a precision of 89% and a recall of 83%. Furthermore, the results of our empirical study show that our framework is effective in filling web forms and recommending values to UI components with the previous user inputs by exploiting usage pattern and mining context information, compared with Firefox Autofill Forms and Google Chrome Autofill Forms. On average, our framework can achieve a precision of 74.5% and a recall of 58% on pre-filling web forms, and a precision of 82.25% and a recall of 68.4% on suggesting values to end-users if the end-users are not satisfied with the pre-filled values.

In the future, we plan to implement the framework as a proof of concept. We also plan to extend the context-awareness of our framework by supporting more contextual data such as user's preferences information extracted from social network activities (*e.g.*, Facebook profile). We want to improve our dataset by collecting more user inputs from different end-users and recruit them conduct a user study to evaluate our framework on pre-filling web forms and suggesting values to end-users.

Acknowledgments

We would like to thank all the IBM researchers at IBM Toronto Laboratory CAS research for their valuable feedback on this research. This research is partially supported by IBM Canada Centres for Advance Studies.

References

[1] Enterprise, a car rental website, https://www.enterprise.com/car_rental/renterinfo.do. Last accesed on Feb 4th, 2014.

[2] Gone Sailing Adventures, a boat rental website, http://www.gonesailingadventures.com/. Last accesed on Aug 22th, 2014.

[3] BOWNIK L., GORKA W., PIASECKI A. (2009) ' Assisted Form Filling ', Engineering the Computer Science and IT. InTech, October 2009. ISBN 978-953-307-012-4.

[4] Mozilla Firefox Add-on Autofill Forms, https://addons.mozilla.org/en-US/firefox/addon/autofill-forms/?src=ss. Last accessed on Feb 4th, 2014.

[5] Google Chrome Autofill Forms, https://support.google.com/chrome/answer/142893?hl=en. Last accesed on Feb 4th, 2014.

[6] HARTMANN M. and MUHLHAUSER M. (2009) 'Context-Aware Form Filling for Web Applications', ICSC' 09. IEEE International Conference on Semantic Computing, 2009, pp. 221-228.

[7] TODA G., CORTEZ E., SILVA A. and MOURA E. (2011) '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.

[8] WINCKLER M., GAITS V., VO D., FIRMENICH S., and ROSSI G. (2011) 'An Approach and Tool Support for Assisting Users to Fill-in Web Forms with Personal Information', in SIGDOC' 11, Proceedings of the 29th ACM international conference on Design of communication, pp. 195-202, October 3-5, 2011.

[9] RUKZIO E., NODA C., LUCA De A, HAMARD J., and COSKUN F. (2008) 'Automatic form filling on mobile devices ', Pervasive Mobile Computing, vol. 4, no. 2, pp. 161-181, 2008.

[10] Ticket Liquidator, an event ticket booking website, http://www.ticketliquidator.com/default.aspx. Last accessed on Aug 22nd, 2014.

[11] Greyhound, a bus ticket booking website, http://www.greyhound.ca/. Last accessed on Aug 22th, 2014.

[12] WANG S., ZOU Y, UPADHYAYA B. and NG J., ' An Intelligent Framework for Auto-filling Web Forms from Different Web Applications ', In Proceedings of 1st International Workshop on Personalized Web Tasking (PWT 2013) on IEEE Services, June 2013 - Santa Clara, California, USA. IEEE Computer Society Press.

[13] DOM Tree, http://www.w3schools.com/htmldom/dom_nodes.asp. Last accessed on Aug 22th, 2014.

[14] UPADHYAYA B., KHOMH F., ZOU Y. (2012) ' Extracting RESTful services from Web applications ', 2012 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pp. 1-4, Taipei, Taiwan 17-19 Dec. 2012.

[15] Ebay, an e-commerce website, www.ebay.com. Last accessed on Aug 22nd, 2014.

[16] Sahi, http://sahi.co.in/. Last accessed on Aug 22nd, 2014.

[17] Kayak, a travel website, http://www.ca.kayak.com/. Last accessed on Aug 22nd, 2014.

[18] WANG J., HAN J. (2004) ' BIDE:Efficient Mining of Frequent Closed Sequences ', Proceedings of the 20th International Conference on Data Engineering, pages 79-90, 2004

[19] MANNING C. D., RAGHAVAN P., and SCHUTZE H. (2008) 'Introduction to information retrieval', Cambridge University Press.

[20] SALTON G., and YANG G. (1973), 'On the specification of term values in automatic indexing', Journal of Documentation, 29(4), 351-372.

[21] HEYER L., KRUGLYAK S., and YOOSEPH S. (1999) 'Exploring Expression Data: Identification and Analysis of Coexpressed Genes'. Genome Res. 1999. 9: 1106-1115, by Cold Spring Harbor Laboratory Press.

[22] WordNet, http://wordnet.princeton.edu/. Last accessed on Feb 4th, 2014.

[23] TEL-8 Query Interfaces, (http://metaquerier.cs.uiuc.edu/repository/datasets/tel-8/. Last accessed on Feb 4th, 2014.

[24] ZHANG Z., HE B., CHANG K. C. (2004) 'Understanding Web Query Interfaces: Best-effort Parsing with Hidden Syntax', In Proceedings of the 2004 ACM SIGMOD Conference (SIGMOD 2004), Paris, France, June 2004.

[25] HE B., CHANG C., and HAN J. (2004) ' Discovering Complex Matchings across Web Query Interfaces: A Correlation Mining Approach '. In Proceedings of the 2004 ACM SIGKDD Conference (KDD 2004) (Full Paper), Seattle, Washington, August 2004.

[26] ARAUJO S., GAO Q., LEONARDI E. , HOUBEN J. (2010) ' Carbon: domain-independent automatic web form filling ', Proceedings of the 10th International Conference on Web Engineering (*ICWE'10), Berlin, Heidelberg, 292-306.*

[27] CHAMBERS R. L., and SKINNER (2003), ' Analysis of Survey Data ', Wiley, ISBN 0-471-89997-9, 2003.

[28] WANG S., UPADHYAYA B., KEIVANLOO I., ZOU Y., NG J., and NG T. (2014) ' Automatic Propagation of User Inputs in Service Composition for End-users ', Proceedings of IEEE Conference on Web Services (ICWS), Anchorage, Alaska, June 27th- July 2nd, 2014.

[29] Chang, C.-H., Kayed, M., Girgis, M. R., and Shaalan, K. F. (2006) ' A survey of web information extraction systems ', IEEE Transactions on Knowledge and Data Engineering, 18(10), 2006, 1411-1427.

[30] WANG S., ZOU Y., UPADHYAYA B., KEIVANLOO I., and NG J. (2014) ' An Empirical Study on Categorizing User Input Parameters for User Inputs Reuse ', Proceedings of the 14th International Conference on Web Engineering, Toulouse, France, July 1-4, 2014.

[31] YIN R. K. (2002) ' Case Study Research: Design and Methods-Third Edition ', 3rd. SAGE Publications, 2002.

[32] RoboForm, http://www.roboform.com/. Last accessed on Jan 25th, 2014.

[33] LastPass. http://www.lastpass.com/. Last accessed on Jan 25th, 2014.

[34] 1Password. https://agilebits.com/. Last Accessed on Jan 25th, 2014.

[35] WANG Y., PENG T., ZUO W., and LI R. (2009) 'Automatic Filling Forms of Deep Web Entries Based on Ontology', Proceedings of the 2009 International Conference on Web Information Systems and Mining (WISM'09). Washington, DC, USA, 376-380.

[36] KHARE R. (2006) 'Microformats: The Next (Small) Thing on the Semantic Web?', IEEE Internet Computing, vol. 10, no. 1, pp. 68-75, January/February, 2006.

[37] HICKSON I. 'HTML Microdata'. http://www.w3.org/TR/microdata. Last accessed on March 25th, 2013.

[38] STYLOS J., MYERS B. A. and FAULRING A. (2004) 'Citrine: providing intelligent copy-and-paste', Proceedings of ACM Symposium on User Interface Software and Technology (UIST), 2004, pp 185-188.

[38] AGRAWAL R. and SRIKANT R. (1994) 'Fast algorithms for mining association rules in large databases', Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pages 487-499, Santiago, Chile, September 1994.

[40] ABOWD G., DEY A., BROWN P., DAVIES N., SMITH M., and STEGGLES P. (1999) ' Towards a Better Understanding of Context and Context-Awareness ', Proceedings of First International Symposium on Handheld and ubiquitous computing, pp. 304-307, Berlin Heidelberg, 1999.

[41] BALDAUF M., DUSTDAR S., and ROSENBERG F. (2007). ' A survey on context-aware systems ', International Journal of Ad Hoc and Ubiquitous Computing, 2(4), 263-277.