

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

Shaohua Wang^{*†‡}
*School of Computing
Queen's University
Kingston, Canada
shaohua@cs.queensu.ca

Ying Zou^{†‡}, Bipin Upadhyaya^{†‡},
†Electrical and Computer Engineering
Queen's University
Kingston, Canada
{ying.zou, bipin.upadhyaya}@queensu.ca

Joanna Ng[‡]
‡CAS Research
IBM Canada Software Laboratory
Markham, Ontario, Canada
jwng@ca.ibm.com

Abstract—Nowadays, people use on-line services to conduct various tasks such as on-line shopping and holiday trip planning using web applications. Generally users are required to enter information into web forms to interact with the web applications. However they often have to type in the same information to different web applications repetitively. It could be a tedious job for a user to fill in a large amount of web forms with the same information. To save users from typing redundant information, it is critical to propagate and pre-fill the user's previous inputs across different web applications. However, existing software and approaches cannot meet this urgent need. In this position paper, we propose an intelligent framework to propagate user's inputs across different web applications. Our framework collects user's inputs and analyzes the patterns of user's usage. Furthermore it detects the changes of user's contexts by extracting user's contextual information from various sources such as a user's calendar. Our framework clusters the user interface (UI) components to form semantic groups of similar UI components based on our proposed clustering approach. Knowing the similarity relation between UI components, the framework can pre-fill the web forms with user's previous inputs. We conduct a preliminary study on effectiveness of our proposed clustering approach. We achieved a precision of 80% and a recall of 87%.

Index Terms—Form Auto-Filling; Clustering; Context-Aware

I. INTRODUCTION

Nowadays Internet is playing an important role in people's life. Users conduct various tasks, such as booking airline tickets, reserving hotel rooms and shopping, using web applications or services. Generally users are required to enter information into web forms which often contain multiple data input fields such as radio buttons to interact with the web applications or services. However, the users often have to repetitively type the same information into different web forms. For example, a user has to input their personal information, such as the ones illustrated in Figure 1, into Enterprise¹ car rental website to book a rental car. When the user signs up the WebMD² health care website, they are also required to enter their personal information as illustrated in Figure 2. The personal information can be identical in both websites. The user has to input the same personal information again into WebMD although the user has entered it in the car rental website. It could be a tedious and repetitive job for a user to fill in a large amount of web forms with multiple input fields. A large amount of information in the various websites

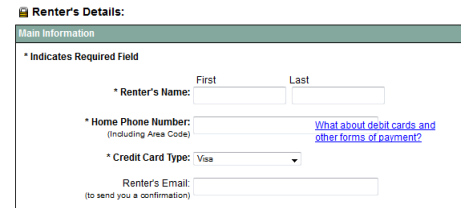


Fig. 1: An example screen shot of a web form requiring user's personal information to reserve a car.

Sign Up

Registering with WebMD is quick and easy... and free! Fill out the information below, and you're on your way to enjoying better information and better health.



Fig. 2: An example screen shot of a web form requiring user's personal information to sign up the website.

can be the same. To save users from repeatedly typing, it is more efficient for users if the information commonly required among different websites can be propagated and pre-filled for the user using the previous inputs across different web applications.

In the recent years, industry and academic worlds have developed several tools and approaches to address the problem. Web browsing software provides web form Auto-filling tools, such as Mozilla Firefox Add-on Autofill Forms [1] and Google Chrome Autofill forms [2], to help users fill in forms. In general, the auto-filling tools record the values entered by a user in a given form and fill the entered values into the same form when the user visits the same website again. The tools also allow users to modify the values manually. Recent academic studies (For example [3][4][5]) have also proposed approaches to help users fill in web forms. M. Winckler et. al. [5] propose an approach to support the exchange of data between personal data and web forms. M. Hartmann et. al. [3] present a novel mapping process for linking contextual information and user interface components to fulfill the form. G. Toda et. al. [4]

1. https://www.enterprise.com/car_rental/renterinfo.do

2. <https://member.webmd.com/register.aspx>

present a probabilistic approach for automatically using data-rich text to fill form-based input interfaces. E.Rukzio et. al [6] 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 these tools and approaches suffer from at least one of the following drawbacks:

- **User Interface (UI) Widget dependent:** Some tools (e.g., Google Chrome Autofill forms [2]) can only fill values into text fields and require exact string matching to form fields.
- **Limited support of collecting and analyzing user’s inputs to detect contexts:** The users have to manually create their profiles containing personal data records and preferences in the tools so that the values of the person profile can be filled into web forms next time. Most of tools, such as Mozilla Firefox Add-on Autofill Forms [1] and Google Chrome Autofill forms [2], lack of a mechanism to analyze user’s inputs to mine the patterns of user’s usage. They cannot detect user’s contextual information from various sources such as a user’s calender, current time, and social networks.
- **No propagation of user’s previous inputs to different web applications used by a user:** A collection of web applications can be linked implicitly by a user’s usage habit. For example, a user usually buys concert tickets from *ticket liquidator*³ and then books bus tickets from *Greyhound*⁴. These two web applications are linked implicitly by a user. Existing tools cannot reuse user’s inputs among these web applications. For example, if the user books two tickets from *ticket liquidator*, the framework should be aware that this user would buy two tickets from a bus transportation company.

In this position paper, we propose an intelligent framework to address the aforementioned limitations. Our goal is to explore the similarity between user’s inputs among different web applications. Understanding the relationships between user’s inputs is a crucial step for automatically propagating user’s previous inputs between different web applications. Our proposed intelligent framework can automatically detect user’s inputs and build user profiles containing personal and preference information, then organize and analyze the inputs to generate the patterns of user’s usage for offering better aid to users for filling in web forms. For example, knowing the patterns of user’s usage, the framework can automatically distinguish different user profiles. The framework should also be context-aware to detect the changes of user’s contexts such as user’s current location dynamically. Our framework supports for exchanging the same information between different User Interface (UI) components such as exchanging the same information presented in different UI components (e.g., drop-down box and text fields). Furthermore the framework can recommend a new list of information in addition to the automatically pre-filled information.

The rest of the paper is organized as follows. Section II

presents an overview of our proposed framework. Section III presents the preliminary results. Section IV summarizes the related literature. Finally, Section V concludes the paper and outlines some avenues for future work.

II. OUR PROPOSED FRAMEWORK

This section presents the overall architecture of our framework as shown in Figure 3. Our framework consists of 6 major components. When a user opens a web page, the *UI Analyzer* parses the web page, extracts the descriptive textual information of the UI components requiring a user action and stores the extracted information into the *UI Components Repository*. The *User Input Collector* collects the user’s inputs and actions (e.g., click, search and submit) and stores the information in the *Input Repository*. The *UI Components Repository* and the *Input Repository* are stored on a user’s local file system. Furthermore the *Input Analyzer* generates the patterns of user’s usage by mining the user’s previous inputs and actions in the *Input Repository*.

The *Similar UI Components Analyzer* analyzes the descriptive information of UI components and clusters them into semantic clusters stored in the *Similar Components Repository* where the UI Components in a cluster can potentially exchange information. The *Contexts Rule Engine* extracts user’s contextual data from the external data sources (e.g., Google calender and Facebook) and analyzes the contexts based on the defined rules. Then it feeds the analyzed contextual information into the *Auto Filler*. The *Auto Filler* pre-fills in web forms with user’s previous inputs and recommends other inputs in case that the pre-filled values are the not ones the user wants. It can pre-fill in not just the text fields, but also other types of UI components such as drop-down boxes

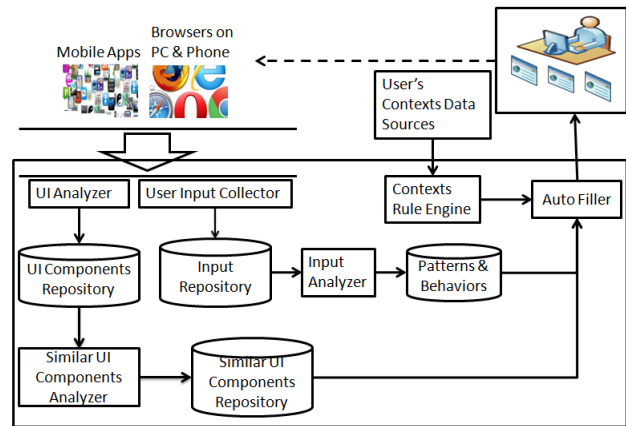


Fig. 3: Overview of our proposed framework for propagating user’s previous inputs across different web applications

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

A. User Input Collector, Input Analyzer and UI Analyzer

To collect user’s information on both PCs and Mobile Phones, we modify and extend an open source tool called

3. <http://www.ticketliquidator.com/default.aspx>

4. <http://www.greyhound.ca/>

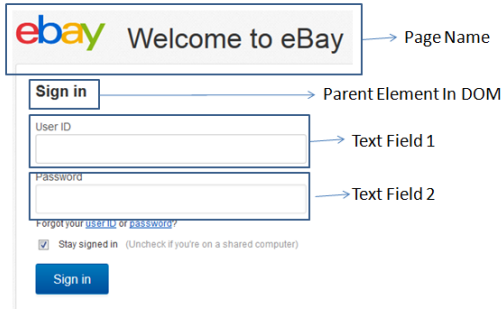


Fig. 4: An example of a HTML page

Sahi⁵ which is used for automating web application testing. 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), and generates a log.

The *User Input Collector* extracts inputs and actions from the generated log from Sahi. By default, the Sahi tool does not include the detailed information (e.g., descriptive information) of a UI component in the log, we extend the tool to extract more information of the UI components related to a user’s action.

The *UI Analyzer* parses the HTML DOM tree [7] of a web page, generates a unique id for every UI component in order to locate it in the *UI Components Repository*, and extracts the descriptive textual information for a UI component. The textual information is used by the *Similar UI Components Analyzer* for clustering similar UI components.

The *Input Analyzer* mines the statistics of user’s usage from the *Input Repository* to capture the user’s usage patterns which can be used by Auto Filler. For example, a user always books a round-trip business class flight ticket from Toronto to Los Angeles for business trips and a round-trip economic class flight ticket from Toronto to Vancouver for personal trips.

The *UI Analyzer* extracts the following information related to a UI component such as the Text Field 1 in Figure 4:

- The name of a web page where the UI component locates, for example, the page name of Text Field 1 in Figure 4.
- The descriptive information of the element in a UI component, such as the values of the attributes of the element such as id, name and text, the value of a label of the element.
- The descriptive information of the *parent element* of the UI component in HTML DOM tree. For example, all the descriptive information of Form 1 in Figure 4 is extracted.
- The type of the UI component (e.g., drop-down box or text fields).
- The unique ids of its neighbor elements which are in the same level as the UI component and share the same parent element of the UI Component. For example, the generated unique id of Text Field 2 in Figure 4 will be extracted. We track ids of the neighbors to make sure that we do not calculate the similarity between two neighbors.

The *UI Analyzer* merges all the descriptive textual information together to construct a bag of words [8] for each component. It stores a record entry for a UI Component in the *UI Components Repository* in the following format: $\langle \text{UniqueID}, \text{a bag of words}, \{\text{Unique Ids of its neighbors}\} \rangle$

B. Similar UI Component Analyzer

The Analyzer clusters the UI components stored in the *UI Components Repository* to form semantic clusters using their textual information. In our clustering approach, we weight each word and a bag of words of a UI component can be represented as a vector of weight values. We use the cosine similarity algorithm [11] to calculate the similarity between two vectors (i.e., two UI components) and Quality Threshold Clustering [10] to cluster similar UI Components.

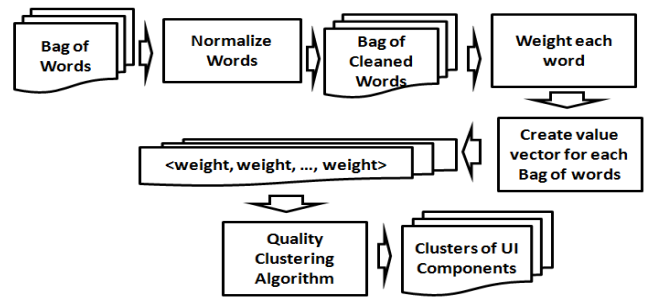


Fig. 5: Overview of our approach for clustering UI components

Figure 5 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 [9] to remove non-English words. The stop words removal, word stemming and lemmatization 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 is 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.

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

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

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

$$\text{weight} = TF \times IDF \quad (3)$$

5. <http://sahi.co.in/>

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

3) Quality Threshold Clustering [10]: 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. The drawback of QT, however, is that it requires more computation resources. We measure the similarity $Sim(C_i, C_j)$ between UI components C_i and C_j using the cosine similarity algorithm [11]. 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 pair of UI components except the ones 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 the components potentially require the same inputs from users.

C. Contexts Rule Engine and Auto Filler

To make it context-aware, the framework detects the changes of user’s contexts and adapts itself to the changes. This contextual information ranges from physical data such as user’s location to “virtual” data such as user’s to-do list. This component extracts the user’s contexts from external data sources, such as a user’s calendar and current location from user’s GPS, and analyzes the contexts based on the predefined rules to command the Auto Filler what to do when the framework detects a user’s context. One challenge is to map the contextual data to the input elements. In most cases, the descriptive text used in user interface differs from the one used to describe the context [3]. For example, the location information in a calendar entry has the label “Location” whereas the location information for the UI in a car rental website is “pick-up”. We plan to adopt the approach proposed by M. Hartmann et. al. [3] for mapping contextual information to UI elements. We combine string-based and semantic similarity measures, and also learn from observing the user’s interactions

The *Auto Filler* consumes the outputs from the Input Analyzer and the Contexts Rule Engine to pre-fill web forms with user’s previous inputs and recommend possible other inputs to users in case that the pre-filled values are not the ones the user wants.

III. EVALUATION

In this section we discuss some of our preliminary evaluation of our approach. We conduct a preliminary study on the effectiveness of our clustering approach.

A. Experiment Setup

We download 40 mobile apps; 20 of them are in the domain of Social and the rest belongs to the domain of Flights Booking. We extract 162 input fields (e.g., text fields, radio button, drop-down boxes) which require a user to enter information.

B. Analysis Approach

To validate our approach, we randomly pick 50 input fields from 162 input fields, cluster them manually and label them to be used for our validation as a golden standard. The 162 input fields are automatically clustered using our clustering approach. We manually identify the labeled 50 input fields among clusters and compare them to our golden standard, then we compute the precision and the recall of our clustering approach using respectively Equation (4) and Equation (6).

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

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

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

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

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.

C. Results

The clustering approach generated 78 clusters in total. Each cluster has 2 input fields on average and the largest one contains 9 fields. Within a cluster, all the components potentially require the same inputs from users, If one of them has a user’s previous input, it is likely that the other components require the same input. Our approach achieves a precision of 80% and a recall of 87%. Matching a label to an element of DOM tree is a challenging task. Some of the extracted UI components we have just simply do not have a meaningful descriptive information. This is the major reason why the precision and recall are not 100%

IV. 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. Web browsing software provides web form Auto-filling tools (e.g., Mozilla Firefox Add-on Autofill Forms [1] and Google Chrome Autofill forms [2]) to help users fill in forms. RoboForm [12] is specialized in password management and provides form auto-filling function. LastPass [13] is an on-line password manager and form filler. 1Password [14] 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 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. In addition, it is context-aware.

Some studies (e.g., [5], [16], [17]) 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 Microformats [20] and Micodata [21]. In [5], M. Winckler et. al. 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 (e.g., [18]) require apriori [19] 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. M. Hartmann et. al. [3] 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.

V. CONCLUSION AND FUTURE WORK

In this position paper, we address the problem of form auto-filling by exploring the relationship between user inputs. We propose an intelligent framework to help users fill in web forms to save them from repeatedly entering the same information. Our framework shares and reuses user's previous inputs by clustering the similar UI components into semantic clusters. Based on our preliminary case study, our clustering algorithm can achieve a precision of 80% and a recall of 87%.

In the future, we plan to continue to implement the framework as a proof of concept. For example, we plan to make our framework understand user's contexts and detect the changes of the contexts dynamically to provide a better web form auto-filling to users. Once the implementation of our framework is done, we will conduct a series of evaluations on our framework. We will apply our clustering approach to more web applications from different domains and calculate precision and recall to measure the effectiveness. We will conduct a user study to evaluate our system to auto-fill web forms and recommend next actions to users.

ACKNOWLEDGMENT

We would like to thank Diana Lau at IBM Toronto Laboratory CAS research for her valuable feedback to this research.

This research is partially supported by IBM Canada Centers for Advance Studies.

REFERENCES

- [1] Autofill Forms - Mozilla Firefox add-on. Available at <https://addons.mozilla.org/en-US/firefox/addon/autofill-forms/?src=ss>. Last Accessed on March 20th, 2013.
- [2] Autofill forms - Google Chrome. Available at <http://support.google.com/chrome/bin/answer.py?hl=en&answer=142893>. Last Accessed on March 20th, 2013.
- [3] Melanie Hartman and Max Muhlhauser. Context-Aware Form Filling for Web Applications. ICSC' 09. IEEE International Conference on Semantic Computing, 2009, pp. 221-228.
- [4] G. Toda, E. Cortez, A. Silva, E. Moura. A Probabilistic Approach for Automatically Filling Form-Based Web Interfaces. The 37th International Conference on Very Large Data Base, August 29th - September 3rd 2011, Seattle, Washington.
- [5] M. Winckler, V. Gaits, D. Vo, S. Firmenich, G. Rossi. 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.
- [6] 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.
- [7] HTML DOM tree: http://www.w3schools.com/html/dom/dom_nodes.asp. Last Accessed on March 25th, 2013.
- [8] Bag-of-words model: http://en.wikipedia.org/wiki/Bag-of-words_model. Last Accessed on March 25th, 2013.
- [9] WordNet: <http://wordnet.princeton.edu/>. Last Accessed on March 25th, 2013.
- [10] L. Heyer, S. Kruglyak, and S. Yooseph. Exploring Expression Data: Identification and Analysis of Coexpressed Genes. Genome Res. 1999. 9: 1106-1115, by Cold Spring Harbor Laboratory Press.
- [11] Cosine Similarity Algorithm: http://en.wikipedia.org/wiki/Cosine_similarity. Last Accessed on March 25th, 2013.
- [12] RoboForm: <http://www.roboform.com/>. Last Accessed on March 25th, 2013.
- [13] LastPass: <http://www.lastpass.com/>. Last Accessed on March 25th, 2013.
- [14] 1Password: <https://agilebits.com/onepassword>. Last Accessed on March 25th, 2013.
- [15] S. Araujo, Q. Gao, E. Leonardi, J. Houben. Carbon: domain-independent automatic web form filling. In Proceedings of the 10th International Conference on Web Engineering (ICWE'10), Berlin, Heidelberg, 292-306.
- [16] L. Bownik, W. Gorka, A. Piasecki. Assisted Form Filling. Engineering the Computer Science and IT. InTech, October 2009. ISBN 978-953-307-012-4.
- [17] Y. Wang, T. Peng, W. Zuo, R. Li. Automatic Filling Forms of Deep Web Entries Based on Ontology. In Proceedings of the 2009 International Conference on Web Information Systems and Mining (WISM'09). Washington, DC, USA, 376-380.
- [18] J. Stylos, B. A. Myers, and A. Faulring. Citrine: providing intelligent copy-and-paste. in Proceedings of UIST, 2004, pp 185-188.
- [19] Apriori: http://en.wikipedia.org/wiki/Apriori_algorithm. Last Accessed on March 25th, 2013.
- [20] R. Khare. Microformats: The Next (Small) Thing on the Semantic Web?. IEEE Internet Computing, vol. 10, no. 1, pp. 68-75, January/February, 2006.
- [21] I. Hickson. HTML Microdata. <http://www.w3.org/TR/microdata>. Last Accessed on March 25th, 2013.