Encapsulating Multi-stepped Web Forms as Web Services

Tobias Vogel, Frank Kaufer, and Felix Naumann

Hasso Plattner Institute, University of Potsdam, Germany
firstname.lastname@hpi.uni-potsdam.de

Abstract. HTML forms are the predominant interface between users and web applications. Many of these applications display a sequence of multiple forms on separate pages, for instance to book a flight or order a DVD. We introduce a method to wrap these multi-stepped forms and offer their individual functionality as a single consolidated Web Service. This Web Service in turn maps input data to the individual forms in the correct order. Such consolidation better enables operation of the forms by applications and provides a simpler interface for human users.

To this end we analyze the HTML code and sample user interaction of each page and infer the internal model of the application. A particular challenge is to map semantically same fields across multiple forms and choose meaningful labels for them. Web Service output is parsed from the resulting HTML page. Experiments on different multi-stepped web forms show the feasibility and usefulness of our approach.

Keywords: deep web, web services, html forms.

1 Multi-stepped Web Forms

The Web has changed several tasks that have been achieved by other means, earlier. Telephone numbers are found on `yellowpages.com`, books can be bought on `amazon.com`, etc. These pages have a common feature, which is the usage of HTML forms to collect user input (names, book titles). Hence, the content of the resulting pages, e.g., the availability of a book, is "hidden" behind these forms. This phenomenon is commonly coined the *Deep Web* or *Hidden Web* [2]. Even more common, results are not shown after filling a simple form, but after having stepped through several subsequent forms. Research on how to exploit the data within the Deep Web itself is already an ongoing research topic [7,10,12,13].

Figure 1 illustrates such a flow of sequential forms (a multi-stepped web form), the New York City public transport[1]. On the start page (1), details such as origin and destination of the trip, but also temporal and route preferences are requested. Depending on the inserted values, one or both of the intermediate pages (2a/2b) come up, asking for additional/corrected values, until finally the resulting page (3) is presented.

[1] `http://lirr42.mta.info/sfweb/faces/index.jspx`

A. Dan, F. Gittler, and F. Toumani (Eds.): ICSOC/ServiceWave 2009, LNCS 6275, pp. 488–497, 2010.

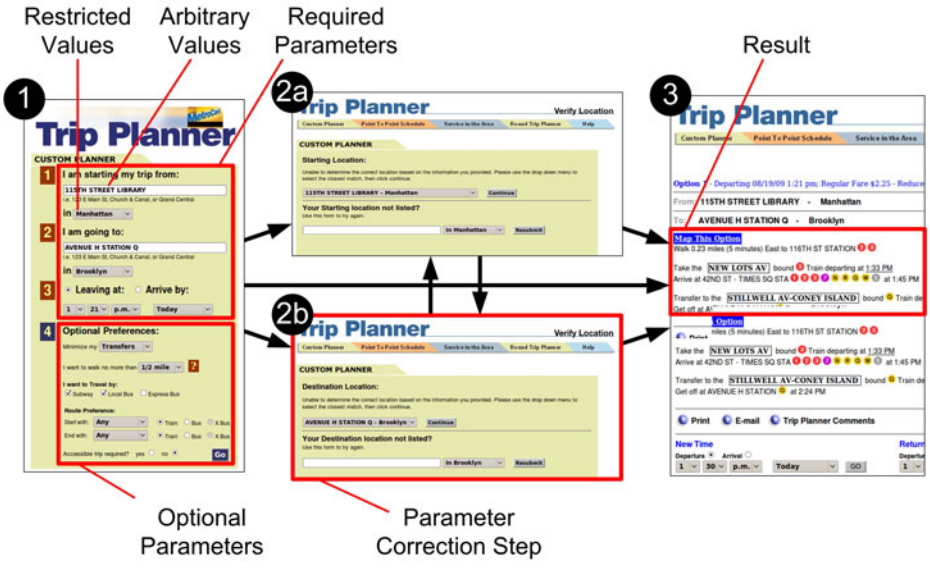

Fig. 1. Example for a common web application (public transportation) and possible user navigation paths through it

1.1 Challenges

Multi-stepped web forms are not easy to analyze, because correspondences between forms in different steps (c.f. Section 2) have to be captured, in contrast to single-stepped web forms. The thereby unified view on the forms is offered as a Web Service. Thus, the resulting Web Service interface has to be a subset of the elements occurring within the forms. This subset has to be defined and the corresponding parameters have to be annotated in a way that the Web Service can be used properly, afterwards. Furthermore, it is not sufficient to create the plain result pages, because these have to be handled again to return a Web Service compliant result.

We put forth two use cases, one for provisioning of human-oriented web applications to computer programs and one for leveraging the use of web applications for humans.

First, web applications wrapped as Web Services enable computer programs to access these applications originally intended and designed for human use, only.

Second, also humans benefit directly from these Deep Web queries by facilitating the user interface. Often, input data forms are spread out over several web pages to reduce the amount of input data per page for the user or to cope with dependencies. However, for re-occurring queries this means unnecessary overhead for the end user, because the user might never decide for different settings, so he does not need to be asked for it each time he uses the application.

When wrapping these applications as Web Services, they can be presented in a much more condensed form, all input fields and controls can be located in one place, if XForms are used to create user interfaces from WSDL descriptions [9].

1.2 Contributions

We make the following contributions:

- We present a technique to match schemas of different web forms to a joint schema, which is offered with a Web Service interface.
- We also show how to retrieve meaningful annotations for these schema elements, which are then used in the Web Service's WSDL description.
- Further, we introduce a method for extracting the result of the aforementioned queries and transforming it into a valid Web Service response.
- The whole prototype system is provided as a web-based application, which proxies the user's interaction with the original page. Since some web applications use session information and user login fields in their forms, these are also considered.

A web-based prototype of the system was implemented and integrated into the PoSR (Potsdam Service Registry) project [1]. PoSR is available at `http://posr.ws/`[2]

This paper is structured as follows: Section 2 goes deeper in the common practice of web applications and explains what we call "background model". Based on that, Section 3 explains the web application analysis process and the heuristics and algorithms used. The results are evaluated in Section 4. Section 5 gives an overview of related research in this field. The contribution is summarized in Section 6.

2 A Model for Multi-stepped Web Forms

2.1 Variables in Web Applications

Web applications make use of variables to save internal states and previous user input, much like other applications. However, since the user interaction takes place in a web browser, these variables are observable during the interaction, e.g., as name attributes of input elements or as parameters in HTTP requests.

Although the complete set of variables is used for the application to achieve the task, not all variables appear in every HTML form of the multi-stepped process. We call this complete set of application variables "background model". It has to be reconstructed as far as possible from the different subsets of variables that occur in each step. This is possible due to the fact, that web applications usually are using a single application engine for the comprised pages and therefore, the variables can be matched.

Figure 2 illustrates such a schema of internal application variables while a formal description is given in Section 2.2. Each row stands for a distinct variable, each column represents a form, in which a variable might or might not appear. In case it appears, it might be renamed, initialized with a value, or given a

[2] The sources are available within the InstantSVC project under `http://instantsvc.sourceforge.net/`

type (corresponding to the type of the form element, e.g., text, hidden, select, check box...). The last column sums up all the variables and thus, comprises the reconstructed model, which is supposed to be the application's background model.

When using the wrapped application via the generated Web Service, all of these variables/values have to be sent to the server, but only a subset of them is specified by the user. For example, variables that are always hidden (`<input type="hidden">`) or those, which remain static values (`<input type=* disabled="disabled">`) are not contained in the Web Service parameter list, but are nevertheless sent by the Web Service.

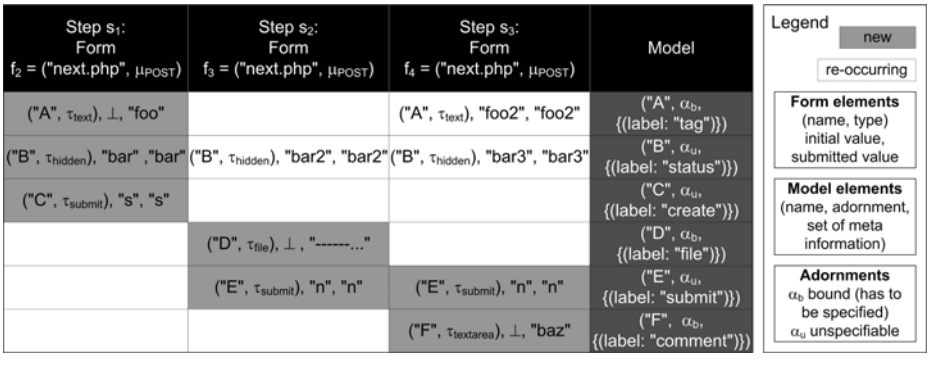

Fig. 2. Illustration of the reconstructed background model of a web application

2.2 Terms and Concepts

Once the schema (Figure 2) is filled, all relevant data are collected. Therefore, the following information has to be captured.

Steps/Pages. A multi-stepped web application comprises several pages. These pages are stored to save the order of proceeding pages.

Forms. Forms contain the controls, the user employs to enter or manipulate data. In general, each page contains one or more forms from which one is implicitly selected by the user when he decides to enter data and to click the "submit" button.

Form elements. There is a large variety of different form elements[3]. Each element has a *name* and a *value*. Form elements can be initialized with values (which can be perceived as default values), but they are also submitted with a value. Both values may differ from each other.

Moreover, there might exist meta information for each form element that contains additional textual descriptions, e.g., `<label>` or `<class>` elements.

[3] Form controls: `http://www.w3.org/TR/html401/interact/forms.html#h-17.2`

Model elements. The application's background model consists of model elements, which are represented as form elements in HTML forms. Thus, a set of one or more corresponding form elements constitutes a model element. Figure 2 shows model elements as rows in the most-right column. It makes sense for the user to assign a value to one model element when using the wrapped application, rather than to assign it to multiple form elements. Section 3.2 shows how the matching works.

3 Analysis of Web Forms

3.1 Phase 1: Monitoring

This section explains how the analysis process is executed, where we follow a supervised approach. Each interaction between the user or—to be more specific—his browser and the web application is monitored. This is technically achieved by using a proxy through which all HTTP requests and responses are routed. After having loaded the web page, the user selects one of possibly several existing forms by filling in values and sends the form. The proxy catches the submission, inspects all form elements of the submitted form (name, description, type, etc.) and stores this information. Afterwards, the request is forwarded to the actual web server and the response is retrieved by the proxy, again. To help the analysis component identify the selected form some information are injected into the web page when retrieved from the web server. When the user submits the form, the proxy does not submit the additional, injected information. Instead it sends a corresponding request with the original form elements/key-value-pairs only. When the user has finished his interaction with the web application, he stops the monitoring process manually.

During the monitoring phase, among others, form elements are inspected. The subsequent matching phase makes use of a multitude of information, which are captured while the monitoring takes place. This information is illustrated in Listing 1.

```
42 <tr>
43   <td>
44     <label for="tel_field">Tel:</label>
45     <input type="text" name="telephone" value="012345..."
46       id="tel_field" class="address_field"
47       disabled="disabled">
48   </td>
49   <td>Insert your telephone number</td>
50 </tr>
```

Listing 1. Example snippet from an address entry form

For each form element, its type and its name (line 45) are saved. These pieces of information are the core information of any form element. There might be an initial value, which becomes the default value of the later model element if this form element is its first participant. Furthermore, other meta information might be available. In the example, an `id` and a `class` attribute are given (line 46). However, better descriptions can be found as visible rendered text as in lines 46 and 53. These descriptions can be offered as labels (line 44) or as text in near proximity (line 49) of the form element [10,12]. Additional meta-information such as in line 47 is also saved. This plays an important role when the WSDL description is created.

3.2 Phase 2: Element Matching

Model elements consist of one or more corresponding form elements. This correspondence relates to similarities in form elements' names, values, or meta information which permit to match those form elements. The resulting mapping contains the model elements that reflect the background model of the application, which are used in the Web Service interface.

The purpose of the matching process is to assign form elements to an existing model element or – if no appropriate model element can be found – to create a new model element for this form element, respectively. This assignment is based on the information gathered in the monitoring phase: form element names ("schema data") and values ("instance data").

For the matching in a given step, all new form elements have to be compared to all existing model elements. Actually, new form elements are compared to the most recently added form elements of the existing model elements. If no appropriate form/model element can be found, a new model element is created.

The focus of this work lies within the retrieval of form structures from web pages and therefore, only a simplistic matching approach is used. Further comparison criteria are possible, e.g., aggregation between different form elements or algebraic manipulations. See Kaljuvee et al. [6] for a comparison between different algorithms.

The comparison comprises two steps. First, a similarity matrix between all possible pairs of form elements and all model elements is created. Note that not model elements, but their last-added form elements are used.

Due to space limitations, we refer to Vogel [11] for details. We used a threshold of 0.6 and the following weights of 0.5/0.75/1.0 for matching values/meta data/-names of two different form elements. However, these weights are preliminary and will be refined in future work.

Given this similarity matrix, a mapping has to be generated in the second step. This is a consistent selection of pairs from this matrix and defines which form elements are assigned to existing model elements or which ones build up new model elements. We employ the *Royal Couples* [8] algorithm.

Once all model elements are updated or created, the Web Service interface definition has to be set up. Not all model elements become input parameters, i.e., are specifiable by the Web Service caller. The ability of model elements

whether and how to be specified is expressed via adornments [14]. Adornments are attributes that state that a variable is (a) mandatory or (b) optional and is arbitrary or fixed-valued (1-out-of-n). See Figure 1 for an intuition. Next to these combinations, a variable can also be unspecifiable, e.g., hidden form fields.

3.3 Phase 3: Result Definition

Web Services do not only need a well-defined input provided via the Web Service interface description, they also have to provide a specified format for their output. Web applications can have many different types of results. In most cases, the result is embedded in the HTML page. Depending on the web application, the result consists of a single or a larger, structured message. XPath provides a means to extract simple or well-structured pieces out of HTML pages.

At the end of the analysis phase, the user is confronted with the result page. He provides a name and a short description to outline the service's purpose. Furthermore, he specifies the XPath expression of the relevant part of the result page by clicking on it or explicitly entering the expression. He also decides whether this expression shall be treated as a string (simple), a list or a table (structured).

In case of a table, each row can be converted into a record of key-value pairs and the Web Service in the end returns an array of these records. All cells are cleaned from markup such as `<b>` or `<font>`, so that the desired content is contained in the records.

4 Evaluation

A preliminary prototype of the system has been implemented. The prototype has been tested on a selection of web applications from different domains. This section shows which types of applications have been monitored and to which extend the generated Web Services are able to wrap these applications.

4.1 Wrapped Web Applications

Streamer Radiative Transfer System. As mentioned in Section 1 this tool converts data between different formats, allowing for specification of a couple of conversation parameters.

Public Transport Advisory. Most public transportation companies provide a lookup service for connections and fares. On `http://ding.eu/`, the potential passenger can select origin and destination of the intended trip. In the next step, he can define the time of the trip and correct possibly miss-spelled input from the first form. Afterwards, the passenger gets a tabular list with matching trips.

Online Shopping. When buying goods online, the shopping cart metaphor comprises a sequence of steps for browsing, selecting, and paying articles, such as at `http://amazon.com/`

URL Shortening. Services to shorten URLs need the address that is to be reduced in size and often an optional short name. They are only single-stepped. A representative for these services is `http://tinyurl.com/`

All of the four scenarios have been put into the monitoring phase. They yielded different, but typical results. In the Streamer case, the interaction with the application is correctly reflected in the captured workflow. The form elements of the settings and the data page are aggregated to a distinct background model. The result in this case is of a simple type: it is the full body of the final page. However, the processing of the provided data takes some time for the web application. This is not reflected inside the Web Service. Thus, the result page is requested before the conversation has finished. To resolve this, the Web Service needs an additional statement to wait a specific amount of time before the last step is executed.

In the public transportation scenario, the input opportunities (origin and destination, timestamp) are successfully wrapped in the Web Service while at the same time respecting their adornments (optional and mandatory parameters). The structure of the tabular result is properly reflected in the Web Service's response. Many of the tested transportation applications have some unspecifiable hidden variables changing their values during the process depending on the provided input. In this case, the created Web Service did not work directly in the generated form, instead the created code had to be adjusted manually to reflect this change.

The Amazon web page actively tries to distinguish between human and machine users and by this, detects the proxy as robot and refuses to fill the shopping cart.

The TinyURL address shortener service can be wrapped without drawbacks. The two fields for the address and the optional alias name are properly recognized and annotated with adornment and description ("url" and "alias").

4.2 Results

The results of the wrapping of four scenario representatives are described in a more general point of view. Generally it is obvious that some practical reasons yet prevent the system from generating proper Web Services respectively prevent the Web Services from working as intended.

- The matching algorithm performs well on the provided information. The Web Service interface contains the expected parameters without duplications or missing parameters. They also obtain helpful descriptions which allow a user familiar with the wrapped application to decide on which values to put in.
- Single-stepped web applications impose a smaller number of barriers to the system. They cannot raise issues such as changing URIs or model elements because these are settled before the user enters any case-specific data. There are a couple of services which wrap these single-stepped applications[4] in RSS

[4] `http://www.dapper.net/`, `http://feed43.com/`, `http://www.feedyes.com/`, `http://ponyfish.com/`

format. With the approach presented here, a standardized interface definition format (WSDL) as well as a standardized messaging format (SOAP) can be used.

This loss of generality might render the approach as impractical for an overall use. However, the system can be taken as a framework to create Web Service stubs. If the monitored user has a developer background, he can employ the generated Web Service code as a foundation for manual adjustments to achieve the originally intended goal. This is necessary for peculiarities such as waiting time (Streamer example) or changes in schema data (Transportation example), described above.

5 Related Work

This work incorporates different research areas. The overall task is to provide means to extract data from the Hidden Web [2,10]. To reveal this data, forms have to be filled with meaningful data [7]. For the sake of scalability, this is usually done automatically. However, there are also supervised approaches [3], which use wrappers to assist web crawlers in their information retrieval process.

Moreover, schema extraction and label recognition is a crucial part of this work. Prior work [6,12] presents sets of methodologies to extract labels for form elements as well as descriptions for semi-structured (tabular) data as it appears on result pages, frequently. This is supported by methods to identify the relevant sections in these pages to make the extraction process more effective [13].

Dury et al. approach a quite similar challenge [5]. They monitor the HTTP traffic between a client and a server to infer the behavior of the web application represented as a finite automaton. The authors trace the application's behavior by causing many client server interactions and applying a rule inference engine to find out the conditions, branches, and guards between the automaton's states.

Kabisch et al. [4] exploit web databases by extracting query interfaces. They analyze web forms regarding their text and form field nodes and arrange them into tree structures. The benefit is that non-leaf nodes provide additional information about the meaning of input parameters.

6 Summary

In this paper, we presented a novel approach to generate wrappers for multi-stepped web applications. These wrappers are implemented as Web Services, which encapsulate the offered functionality. The HTML forms used as graphical user interface of the applications are analyzed and corresponding elements, contained in these forms, are matched. With this, it is possible to create distinct, comprehensively described parameters for the Web Service. Tabular application results are parsed into objects.

Experiments show that it is indeed feasible to encapsulate multi-stepped applications. The matching algorithm finds corresponding form elements in different

steps and reconstructs the application's background model. However, in practice, the resulting Web Services are not usable out-of-the-box in every case. E.g., delays, redirects, or dynamic `action` attributes have to be overcome manually. We plan to enhance our prototype to handle these cases, too.

We see more future work in supporting different work-flows within the application. Currently, only the monitored process is available. If there are deviations, for instance resulting from specific input data, this should be recognized and the Web Service call should react appropriately. Furthermore, the result specification can be facilitated by discovering the key parts of the result page [13] or using more sophisticated label assignment algorithms [4,12].

References

1. AbuJarour, M., Craculeac, M., Menge, F., Vogel, T., Schwarz, J.-F.: PoSR: A comprehensive System for Aggregating and Using Web Services. In: International Conference on Web Services (2009)
2. Bergman, M.K.: The Deep Web: Surfacing Hidden Value. Journal of Electronic Publishing (2001)
3. Carme, J., Ceresna, M., Frölich, O., Gottlob, G., Hassan, T., Herzog, M., Holzinger, W., Krüpl, B.: The Lixto Project – Exploring New Frontiers of Web Data Extraction. In: British National Conference on Databases (2006)
4. Dragut, E.C., Kabisch, T., Yu, C., Leser, U.: A Hierarchical Approach to Model Web Query Interfaces for Web Source Integration. In: Very Large Databases
5. Hallal, H.H., Dury, A., Petrenko, A.: Inferring Behavioural Models from Traces of Business Applications. In: International Conference on Web Services (2009)
6. Kaljuvee, O., Buyukkokten, O., Garcia-Molina, H., Paepcke, A.: Efficient Web Form Entry on PDAs. In: World Wide Web Conference (2001)
7. Madhavan, J., Ko, D., Kot, L., Ganapathyand, V., Rasmussen, A., Halevy, A.: Google's Deep Web Crawl. In: Proc. VLDB Endowment (2008)
8. Marie, A., Gal, A.: On the Stable Marriage of Maximum Weight Royal Couples. In: Workshop on Information Integration on the Web (2007)
9. Menge, F.: Generation of User Interfaces for Service Compositions. Master's thesis, Hasso Plattner Institute at the University of Potsdam (2009)
10. Raghavan, S., Molina, H.G.: Crawling the Hidden Web. In: International Conference on Very Large Databases (2001)
11. Vogel, T.: Generierung von Web Services zur Kapselung mehrstufiger Webformulare. Master's thesis, Hasso Plattner Institute at the University of Potsdam (2009)
12. Wang, J., Lochovsky, F.H.: Data Extraction and Label Assignment for Web Databases. In: International Conference on World Wide Web (2003)
13. Wang, J., Lochovsky, F.H.: Data-rich Section Extraction from HTML pages. In: International Conference on Web Information Systems Engineering (2003)
14. Yerneni, R., Li, C., Garcia-Molina, H., Ullman, J.D.: Computing Capabilities of Mediators. In: International Conference on Management of Data (1999)