Automating the Extraction of Data Behind Web Forms

A Thesis Proposal
Presented to the
Department of Computer Science
Brigham Young University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Sai Ho Yau
June 22, 2000

I Introduction

With the growing trend of using databases and forms to provide information through Web pages, a significant portion of the information on the Web can now only be obtained if a user fills in a form-like Web page which acts as an interface between the user and the serving database. From the server point of view, this paradigm provides better information management and a greater variety of ways to display information. From the user point of view, this paradigm increases flexibility and control over what data is retrieved and displayed. From a Web crawler's point of view, however, this paradigm makes it difficult to extract the data behind the form interface automatically. This automation is desirable (1) when we wish to have automated agents search for particular items of information, (2) when we wish to wrap a site for higher level query, and (3) when we wish to extract and integrate information from different sites.

There are many ways to design Web forms, and dealing with all the possibilities is not easy. Web form layouts have different combinations of form fields such as radio buttons, checkboxes, selection lists, and text boxes. Figures 1 and 2 show two typical Web form layouts for automobile classified advertisements. These two forms include selection lists and text boxes. Besides having various form fields, some Web pages lead to other forms for further inquiry. When a user enters a zip code in Figure 2, Figure 3 is returned for further processing. Occasionally, returned Web documents include both retrieved data and a form for further processing.

Besides data and forms for further processing, returned pages might return error messages. In case of unsuccessful retrieval, some error messages are easy to recognize automatically, such as the HTTP error message in Figure 4. Other error messages

Figure 1: Typical Web form from an automobile advertisement Web site

Figure 2: Web form from a different car advertisement Web site

are difficult to recognize automatically, such as the embedded error messages within the page on which the form appeared in Figure 5. Users can usually understand these embedded messages, but automated understanding is difficult.

Sometimes all the data behind a form can be retrieved with a single query. At other times, the data must be obtained piecemeal using multiple queries with different form field settings. When data is obtained piecemeal, we may retrieve duplicate information. We would like to eliminate duplicate information, leaving only a clean set of retrieved data.

Figure 3: Resulting form after the form in Figure 2 is submitted

Figure 4: Error message encompassing an entire page

Figure 5: Error message embedded within a page

All these issues present difficulties for automation. How does a system recognize a form and form fields within it? How can a system automatically fill in the fields of a form and submit it? How can a system deal with retrieved data, duplicate information, possible error messages or error notification pages, and embedded Web forms inside retrieved documents? In this thesis, we tackle these problems and make automated Web form filling possible.

No other system we are aware of attempts to retrieve data behind forms as we do. Other systems have been built that automatically fill in forms and submit them [e.g. DEW97, EBC96, ECo99]. These other systems, however, just act as a tool to provide a user's information to supported target sites by automatically filling in compatible Web forms [EBC96, ECo99], or only focus on specific items of interest [DEW97], rather than on the more general goal of gathering the data available from a target site to be used for further data extraction.

II Thesis Statement

This thesis proposes a system that automatically extracts data behind a Web form. The system fills in HTML forms automatically, retrieves data, and eliminates duplicates.

III Methods

Our approach strives to retrieve the data behind a Web form. The system we intend to build will recognize a form and form fields within it, automatically fill in and submit the form, and deal with retrieved data, duplicate information and error messages.

Recognizing a Web form and its fields is the first step. A standard HTML Web form consists of form tags—a start form tag <form> and an end form tag </form> — within which the form fields reside [Dar99, Sta96]. Form fields may include radio buttons, check boxes, selection lists, and text boxes. Figure 6 shows the source for an HTML form for musical instrument classified ads. The first and last lines contain the enclosing form tags. The <select> tag at the end of the 8th line opens a selection list, and the </select> tag on the 17th line closes it. There are seven items on the selection list.

Our system will parse a Web document and recognize whether it contains a form by detecting the presence of form tags. If form tags exist, the system will construct an array of objects based on the fields of the form. Information to be collected includes form field names, their types (e.g. radio, checkbox, text), and in some cases, their values.

Figure 7 shows an excerpt of the system's marked-up printout of its internal representation of the Web form in Figure 6. The internal representation includes the source URL, the action path where the form is sent for processing, the number of fields and the details for each field. The fields are *win2_Elem_name_0*, *win2_Elem_name_1*,…, and *win2_Elem_name_7*, which respectively have the names *category*, *manuf*, *model*, *year*, *condition*, *sort_by*, *submit_search*. Notice that the default value settings for most of the fields are blank, which means not restricted. Only the value under *sort_by*, which is

```
(1)     <form action="/cgi-bin/umg/search.cgi" method="POST"
(2)     enctype="x-www-form-encoded">
(3)
(4)       <div align="center"><center><table border="0">
(5)         <tr>
(6)               <td><font face="Arial"><strong>Category</strong></font></td>
(7)                <td><font
(8)               face="Comic Sans MS, Verdana, Arial, Helvetica, sans-serif"><select
(9)               name="category" size="1">
(10)                <option selected value="">all categories</option>
(11)                <option value="accessories">accessories</option>
(12)                <option value="acoustic">acoustic (i.e. &quot;violins&quot;)</option>
(13)                <option value="drums">drums</option>
(14)                <option value="guitars">guitars</option>
(15)                <option value="keyboards">keyboards</option>
(16)                <option value="studio/stage">studio/stage</option>
(17)              </select> </font></td>
(18)              <td><font size="1" face="Arial">choose one</font></td>
(19)       </tr>
(20)       <tr>
(21)              <td><font face="Arial"><strong>Manufacturer</strong></font></td>
(22)              <td><font
(23)              face="Comic Sans MS, Verdana, Arial, Helvetica, sans-serif"><input
(24)              type="text" size="30" name="manuf"> </font></td>
(25)              <td><font size="1" face="Arial">example </font><font
(26)              size="3" face="Arial">Gibson</font></td>
(27)       </tr>
(28)       <tr>
 …                        ...
(32)              <input type="text" size="30" name="model">
 …                        ...
(56)       </tr>
(57)       </table>
(58)       </center></div><p align="right"><font
(59)              face="Comic Sans MS, Verdana, Arial, Helvetica, sans-serif">Sort
(60)              results in alphabetical order of <select name="sort_by"
(61)              size="1">
(62)                <option selected value="1">Category</option>
(63)                <option value="2">Manufacturer</option>
 …                        …
(66)              </select><br>
(67)     </font></p>
(68)     <div align="right"><table border="0">
(69)       <tr>
(70)              <td><input type="reset" value="Clear Form"></font></td>
(71)              <td><input type="submit" name="submit_search" value="SEARCH"></td>
(72)       </tr>
(73)     </table>
(74)     </div>
(75)   </form>
```

Figure 6: Excerpt of the source code of a Web form

```
Domain_Path: http://www.usedmusicgear.com
win2_form_action: /cgi-bin/umg/search.cgi
win2_Elem_length_0: 8

win2_Elem_name_0: category
win2_Elem_type_0: select-one
win2_Elem_value_0:
win2_Elem_option_length: 7
  win2_Elem_option_0_0:                text: all categories
  win2_Elem_option_0_1: accessories;   text: accessories
  win2_Elem_option_0_2: acoustic;      text: acoustic (i.e. “violins”)
  win2_Elem_option_0_3: drums;         text: drums
  win2_Elem_option_0_4: guitars;       text: guitars
  win2_Elem_option_0_5: keyboards;     text: keyboards
  win2_Elem_option_0_6: studio/stage;  text: studio/stage
win2_Elem_name_1: manuf
win2_Elem_type_1: text
win2_Elem_value_1:
win2_Elem_name_2: model
win2_Elem_type_2: text
win2_Elem_value_2:
win2_Elem_name_3: year
win2_Elem_type_3: text
win2_Elem_value_3:
win2_Elem_name_4: condition
win2_Elem_type_4: text
win2_Elem_value_4:
win2_Elem_name_5: sort_by
win2_Elem_type_5: select-one
win2_Elem_value_5: 1
win2_Elem_option_length: 4
  win2_Elem_option_5_0: 1;    text: Category
  win2_Elem_option_5_1: 2;    text: Manufacturer
  win2_Elem_option_5_2: 3;    text: Model
  win2_Elem_option_5_3: 16;   text: Condition
win2_Elem_name_6:
win2_Elem_type_6: reset
win2_Elem_value_6: Clear Form
win2_Elem_name_7: submit_search
win2_Elem_type_7: submit
win2_Elem_value_7: SEARCH
```

Figure 7: Excerpt of our internal representation of a Web form

a selection list, has a non-blank default, namely 1, which means that the result will be sorted by category.

Based on the default setting and the standards of the HTML structure, we can construct the query:

http://www.usedmusicgear.com/cgi-bin/umg/search.cgi?category=&manuf=&model=&year=&condition=&sort_by=1&submit_search=SEARCH

This query fills in the form automatically and sends it back to the target site for a response. Figure 8 shows the returned page for this query.

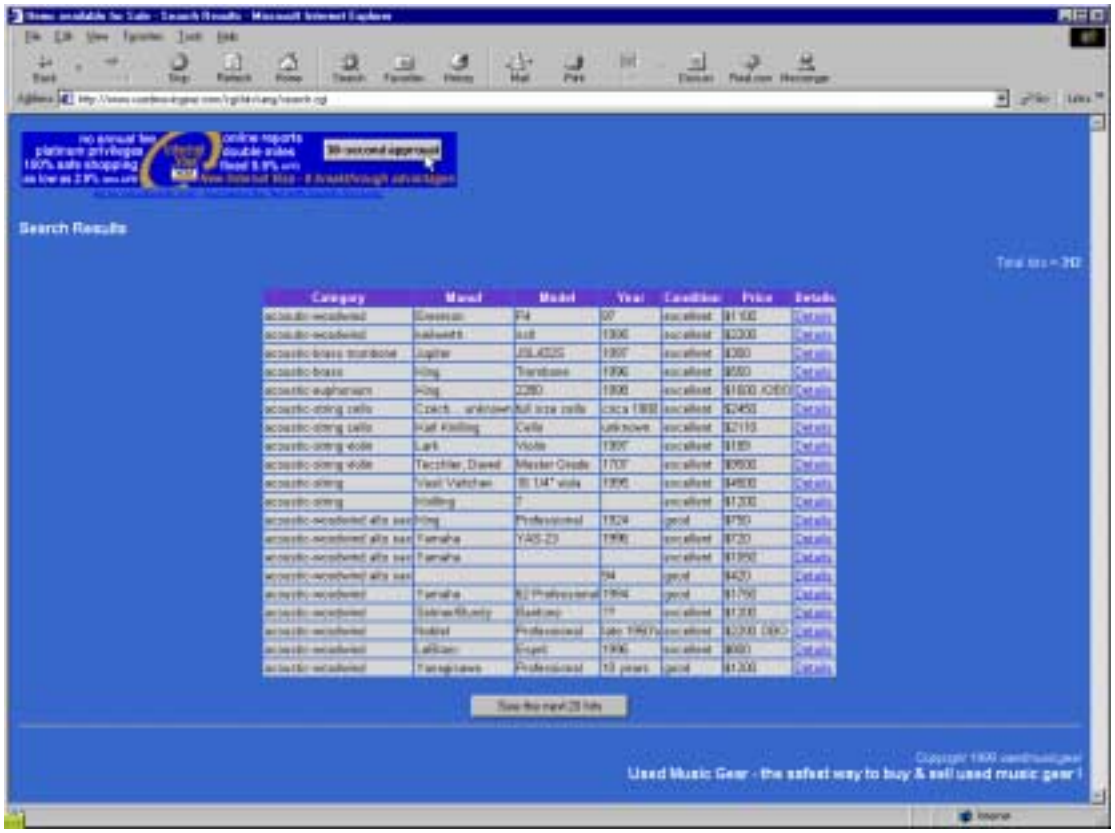

Figure 8: Returned page containing retrieved data based on the search query

We can construct other queries by selecting various combinations of selection-list values, radio-button settings, and check-box selections. Usually, however, it is not

reasonable to fill in text boxes automatically. Our system will allow a user to provide values for text boxes, but will not require that values be provided. For forms with text boxes, our system will only submit queries that have no entries for text boxes or that have user-supplied text-box values. The text boxes in the query that returned the results in Figure 8, for example, were all empty.

When our system submits a form for processing, seven different results are possible. (1) The returned page will contain all the data behind the form. (2) The returned page may contain data, but not show all the data for the query in a single page. Instead, there may be a "next" button leading to another page of data, such as the "See the next 20 hits" button in Figure 8. In this case, the system will automatically gather all the data on all "next" pages into a single query result. (3) The query might return data, but only part of the data behind the form because the query is just one of many possible combinations of the form fields. (4) The query may return a page that not only contains data, but also contains the original form. (5) The query may return a page that has another different form to fill in. (6) The query might return an error message or an error notification page, stating that certain text fields are required to be filled in, or simply a message stating that there is no record found for that submitted query. (7) Some other error cases might involve a server being down, an unexpected failure of a network connection, or some other HTTP errors.

Despite the fact that we have many possible responses to a query, we proceed in only one way (except for case 7 where we immediately stop and report any HTTP error.) We proceed in two phases—a sampling phase and an exhaustive phase. In the sampling phase, we first submit the form with default settings and then we randomly select form-field settings and repeatedly submit the form. Altogether we submit the form n times in the sampling phase. We set n so that the probability is less than p percent that new

information will be retrieved with additional submissions when no new information has been retrieved from the *n*-1 submissions after the first. We let *p* be user defined, typically somewhere around 5 %. If no new information is retrieved after the *n*-1 submissions beyond the first, we stop and report the results to the user. If new information is retrieved, we estimate the total amount of information likely to be returned and the time it is likely to take to return it. If the totals are above a preset threshold, we report this to the user and return the results of the sampling plus the estimate about how much data remains to be retrieved as well as how much time it is likely to take to retrieve it. If the totals are below the preset threshold or if the user makes a special request, we enter the exhaustive phase and submit the form for all remaining form-field settings and return the results to the user.

When we retrieve data, we must merge this data with already retrieved data. Our merge is based on finding duplicate chunks of information [SCa00, Cca00], discarding these chunks and keeping only the new information. After all processing takes place, we can send the fully merged data downstream for further analysis and extraction [ECJ99].

IV Contribution to Computer Science

Our approach enhances the effectiveness of the data-extraction process by retrieving data from Web documents that have form interfaces to their data.

V Delimitations of the Thesis

This thesis will not do the following:

- Deal with a Web form unless it is in a standard form format.
- Automate form filling for required text fields.
- Load a database with extracted information (but will send data to downstream processes that do extract and load the information in a database.)
- Check the Web page for applicability with respect to an ontology.
- Follow hyperlinks to other pages.
- Handle forms that do not support passing parameters via the URL.
- Process pages with multiple forms.
- Process forms that span multiple pages.

VI Thesis Outline

1. Introduction
   - 1.1 The Problem
   - 1.2 Thesis Organization

   Estimated Length: 3 pages
2. Automated Form Filling
   - 2.1 Input Queries for Web Form Document
   - 2.2 Details of Query Submission

   Estimated Length: 15 pages
3. Processing Retrieved Documents
   - 3.1 Strategy for Handling Retrieved Documents
   - 3.2 Duplicate Record Filtering

   Estimated Length: 25 pages
4. Experimental Analysis and Results

   Estimated Length: 5 pages
5. Related Work

   Estimated Length: 2 pages
6. Conclusions and Future Work

   Estimated Length: 4 pages

VII Thesis Schedule

A tentative schedule of this thesis is as follows:

Literature Search and Reading	August - September, December 1999
	January - February 2000
Chapter 2	June, 2000
Chapter 3	July, 2000
Chapter 1 & 4	August, 2000
Chapter 5 & 6	September, 2000
Thesis Revision and Defense	October, 2000

VIII Bibliography

[CCa00] Wendy R. Chen, Douglas M. Campbell, "A Sentence Boundary Detection System," A Master degree of Science thesis, Brigham Young University, Provo, Utah, 2000.

This thesis develops a sentence boundary detection system for Natural Language Processing tasks. Within the sentence boundary, duplicate data can be detected. It is punctuation-rule based, highly accurate (accuracy rate of 99.8%), highly efficient (about 50 pages per second) and easily modifiable.

[Dar99] Rick Darnell, et al., HTML 4 Unleashed, Indianapolis, Ind.: Sams, 1999.

This book describes the structures, syntax, functionality and features of HTML version 4. Examples of the use of HTML are illustrated.

[DEW97] Robert B. Doorenbos, Oren Etzioni, Daniel S. Weld, "A Scalable Comparison-Shopping Agent for the World-Wide Web," in Proceedings of the First International Confence on Autonomous Agents, 1997.

This paper investigates the issues of (1) the extent an agent can understand information published at Web sites, (2) whether the agent's understanding is sufficient to provide genuinely useful assistance to users, (3) the ability of an agent to automatically extract information from unfamiliar Web sites, and (4) the aspects of the Web that facilitate this competence. A case study is presented using ShopBot, a domain-independent comparison-shopping agent. This paper also addresses the problems and ways to find and fill in a proper form automatically.

[EBC96] ebCARD homepage: http://www.patils.com/

This site provides an online form called LiveFORM™ that users can automatically fill out using their electronic business card, ebCARD™, stored on their computer.

[ECJ99] D.W. Embley, D.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.-K. Ng, R.D. Smith, "Conceptual-model-based data extraction from multiple-record Web pages," Data & Knowledge Engineering 31 (1999) 227-251.

This paper introduces a conceptual-modeling approach to extract and structure data. The approach is based on an ontology—a conceptual model instance—that describes the data of interest, including relationships, lexical appearance, and context keywords. By parsing the ontology, the system can automatically produce a database scheme and recognizers for constants and keywords, and then invoke routines to recognize and extract data from unstructured documents and

structure it according to the generated database scheme. This paper also introduces algorithms to discover the record boundary within a Web document.

[ECo99] eCode.com homepage: http://www.eCode.com/

This site is an online identity management service providing its Web form-filling utility, AutoFiller™, that allows Internet users to instantly populate any online form (referring to registration forms) with their contact information.

[SCa00] Randy D. Smith, Douglas M. Campbell, "Copy Detection System for Digital Documents," A Master degree of Science thesis, Brigham Young University, Provo, Utah, 2000.

This thesis develops a copy detection system that gives overlap information at the sentence level. The system does not deteriorate in accuracy due to variations in document size or genre.

[Sta96] William R. Stanek, Web Publishing Unleashed: HTML, CGI, SGML, VRML, Java, Indianapolis, IN : Sams.net, 1996.

This book covers most of the popular Web publishing languages and illustrates the power of those languages with practical examples. It also shows the similarity as well as the differences among those languages on features, structures, and functionality.

IX Artifacts

The program that implements the retrieval of data behind Web forms using the proposed approach will be written in PHP, JavaScript and Java on a Solaris (Intel) server.

X Signatures

This proposal, by Sai Ho Yau is accepted in its present form by the Department of Computer Science of Brigham Young University as satisfying the proposal requirement for the degree of Master of Science.

_____
David W. Embley, Committee Chairman

_____
Stephen W. Liddle, Committee Member

_____
Aurel D. Cornell, Committee Member

_____
J. Kelly Flanagan, Graduate Coordinator