



PROCEEDINGS OF THE
11th ANNUAL CONFERENCE
ON WORLD WIDE WEB APPLICATIONS

2-4 September 2009
Port Elizabeth
South Africa

Editor:
P.A. van Brakel

Publisher:
Cape Peninsula University of Technology
PO Box 652
Cape Town
8000

Proceedings published at
<http://www.zaw3.co.za>

ISBN: 978-0-620-45215-1

TO WHOM IT MAY CONCERN

The full papers were refereed by a double-blind reviewing process according to South Africa's Department of Education (DoE) refereeing standards. Papers were reviewed according to the following criteria:

- Relevancy of the subject to Web applications
- Explanation of the research problem & investigative questions
- Quality of the literature analysis
- Appropriateness of the research method(s)
- Adequacy of the evidence (findings) presented in the paper
- Standardised referencing style.

The following reviewers took part in the process of evaluating the full papers of the 11th Annual Conference on World Wide Web Applications, held from 2 tot 4 September 2009 in Port Elizabeth:

Prof J Brits
Faculty of Information Sciences (Dean)
University of Wisconsin
Milwaukee
USA

Prof T Carr
Centre for Higher Education Development
University of Cape Town
Cape Town

Prof J Cronjé
Faculty of Informatics and Design (Dean)
Cape Peninsula University of Technology
Cape Town

Mr Craig de Beer
Web Management Services
Massey University
Palmerstone
New Zealand

Prof M Erasmus
Management Information Systems
Erasmus Associates
Lynnwood
Pretoria

Dr AM El-Sobky
Educational Sciences
RITSEC
Cairo
Egypt

Dr MWH Labour
Laboratory of the Sciences of Communication
University of Valenciennes and Hainaut Cambrésis
Valenciennes Cedex
France

Prof S Mutula
Department of Information Science
University of Botswana
Botswana

Dr David Raitt
European Space Agency
The Hague
The Netherlands

Prof A Singh
Department of Business Information Systems
University of KwaZulu-Natal
Durban

Prof P Weimann
Economics and Social Sciences
University of Applied Sciences
Berlin
Germany

Further enquiries:

Prof PA van Brakel
Conference Chair: Annual Conference on WWW Applications
Cape Peninsula University of Technology
Cape Town
+27 21 469 1015 (landline)
+27 82 966 0789 (mobile)

Integrated query of the Hidden Web

S. Berman
Computer Science Department, University of Cape Town
South Africa
sonia@cs.uct.ac.za

M. Kamkuemah
Computer Science Department, University of Cape Town
South Africa
mkamkuem@cs.uct.ac.za

J. Muntunemuine
Computer Science Department, University of Cape Town
South Africa
jmuntune@cs.uct.ac.za

Abstract:

There is a need for software that can access multiple Websites through a single, common interface. This would allow users, for example, to compare flights for a particular trip across all relevant airline sites by posing a single query. This paper investigates automating this process in the case of airline databases hidden behind the Web (the so-called Deep Web or Hidden Web). We first constructed a prototype for integrated query of a handful of pre-determined airline sites. This proved useful in detecting commonalities and differences in the sites, and in selecting the most suitable technologies for working with multiple forms. A generic system was then designed and components of the prototype incrementally replaced by domain-specific tools able to handle arbitrary airline sites. Our results were promising as regards result interpretation, with 89% of response pages successfully handled. However query formulation presented many problems, with only 39% of query interfaces automatically interpreted correctly, and even fewer amenable to automated query propagation. We conclude that integrated access to the Hidden Web is considerably more challenging than crawling the Surface Web, and that domain-specific systems are a promising approach to full automation.

Keywords: Hidden Web, Deep Web, query form, result extraction.

1. Introduction

With an ever-increasing amount of data on the Web, consumers and providers become more concerned about how efficiently the data can be accessed. Traditional search engines are only able to search a portion of the Web, called the Publicly Indexable Web (PIW) or Surface Web. It is useful to have databases generate Web pages dynamically. Often, such pages are accessible only through an HTML form that invokes a CGI request to a web server. The Hidden Web (or Deep Web) refers to databases accessible through query interfaces on the World Wide Web. The Hidden Web is estimated to be about 400-550 times greater than the information found on the PIW (Bergman 2001).

In this paper we explore querying multiple Hidden Web databases automatically using a generic but domain-specific tool. Online systems such as Travel Start can query 5 or 6 airlines at the same time but are not generic. We begin with this approach, searching 4

airlines offering flights in Africa, and then investigate expanding this to a general solution able to deal with arbitrary airlines. Whereas existing approaches are semi-automated, by focusing on a specific domain our long-term goal is a system able to *automatically*:

1. recognize and interpret query interfaces to Hidden Web databases,
2. create a query form comprising common elements from those interfaces,
3. translate and submit queries to the individual data sources appropriately and
4. present all returned results in consolidated form.

This will reduce the amount of searching time and simplify the way users go about finding information hidden in multiple databases, freeing them from entering details separately into each individual form. Otherwise finding the relevant websites, querying these databases and studying their results is tedious and time consuming.

In the specific context of airlines on the Hidden Web, we studied questions such as: What proportion of the Deep Web is a generic solution likely to handle correctly? What tools are useful in building such a system? What Internet problems is an integrated query tool likely to encounter? What other problems will need to be handled? How varied are form labels within a domain – will analysis of a few form interfaces suffice to interpret labels on all interfaces in this domain? How much developer effort is required to learn how any given query interface operates? What kinds of query forms lend themselves to integrated query tools and what kinds make this difficult or impossible?

2. Overview

2.1 System Architecture

Our approach comprises two main subsystems – the preparatory and the runtime systems. The preparatory system is run first. Its role is to discover hidden websites in the airline domain and analyse the format of their query pages and of their result pages, and store the findings in a database for use by the runtime system. This database thus records, for each Hidden Web airline site: general information such as the URL of the site; the location and meaning of each form element on its query page; the location and meaning of each returned value on its result pages; and information on how to submit a query to the hidden database. The runtime system obtains input from a single unified query form; it submits queries to each Hidden Web database, and merges results received based on data in the preparatory databases as shown in figure 1. Since interpretation of form and results pages is difficult, it is essential that this step is factored out into a separate system, making the runtime environment as efficient as possible. Both preparatory and runtime components comprise two subsystems, one handling query forms and the other result pages.

2.2 Query formulation

Query formulation requires creating a unified interface (which we call the integrated interface), recognising web pages that contain query forms in the airlines domain, locating the relevant elements on these Hidden Web interfaces, semantically matching attributes of these forms with those of the integrated interface, and submitting user queries to the Hidden Web databases accordingly.

The unified interface must contain common attributes found on query interfaces in the domain. The form was designed manually based on human inspection of the 4 sample airlines used in the initial system (viz. SAA, British Airways, Air Kenya and Air Namibia). It was considered unlikely that there would be any element common to airline queries worldwide that did not appear on these four websites.

In order for the system to access content hidden behind query forms, it should be able to automatically parse, process and interact with form-based query interfaces. Query interfaces, even those belonging to the same domain, are very different, and therefore it is hard to design a general form-filling method to provide input to all these query interfaces. Airline website recognition focuses on locating sections of web pages that contain airline forms. Once identified, relevant attributes are selected. These HTML attributes are then semantically mapped to those of the integrated query interface. The runtime system must translate the user-defined query into URL queries specific to each query interface recognized by the preparatory system.

2.3 Result integration

Result extraction identifies and extracts the pure results from the response pages returned by Web databases. Its role is to parse the response pages and then interpret extracted values in order to map them onto a pre-defined common template and store them in a database accordingly. While simply displaying each result in a separate tab as it is received would allow users to view results sooner, intermediate storage offers several advantages. It enables results from all the airlines to be displayed in one list, which is easier to work with than continually switching between tabbed panes – particularly as there is enormous variety in terms of structure and presentation across airlines. It also allows results to be sorted and queried, thereby facilitating comparison shopping.

There are several problems that need to be solved when interpreting and consolidating results: extracting the relevant pieces of data from these pages; distilling the data and improving its structure; managing heterogeneity; and merging data into a consolidated schema. In addition, the websites contain two types of HTML pages: data pages and navigational pages. Moreover, instances of duplicated data were discovered, as well as malformed HTML.

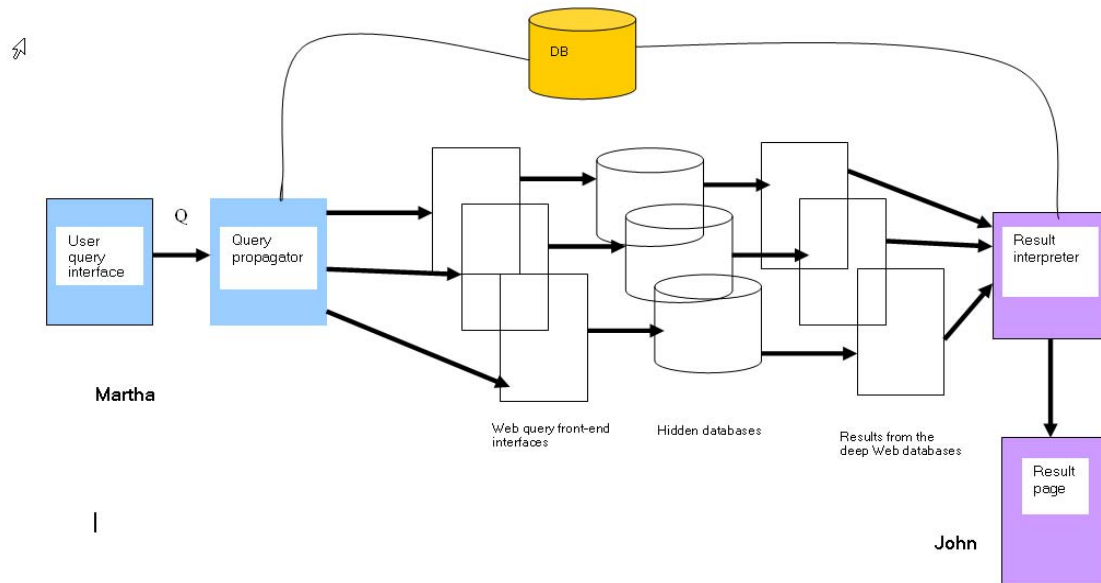


Figure 1: Overview of the Runtime System which uses the database from the Preparatory System.

3. Query Processing

The Query Processing system includes Form Analysis and Query Formulation.

3.1 Parsing query forms

The system was implemented in Netbeans 6.0.1, using the Java programming language. In order to analyze query interfaces, the Jericho HTML Parser (Jericho 2008) was chosen.

Deep Web databases do not publish their schemas and their contents are hard to retrieve. Query form interfaces are usually represented as HTML forms. The Jericho Parser picked up most of the forms. Any of the HTML form elements (i.e. input types: text box, selection list, text area, radio button or checkbox; as well as hidden input tags) have a domain as well as a set of values that can be entered onto the form. Most of the elements have an associated descriptive text called a label. All these features are automatically extracted by the Jericho Parser. A form is identified by the opening and closing form tags. The Jericho Parser recognizes the form tags and picks up the name and value associated with each form element.

Not all forms on a webpage are interfaces to hidden databases. In the airlines domain, the preparatory system also detected forms that do not represent queries, such as login forms.

After parsing four airline interfaces the system appeared sufficiently robust to handle arbitrary Hidden Web airline forms.

3.2 Interpreting form elements and submitting queries

The system is built and deployed in Netbeans as a Web Application Archive (WAR) file. WAR files are collections of web resources such as Java Server Pages, HTML pages, servlets and java programs bundled together and can be distributed and deployed on any platform to a web server. The user interface was developed using HTML. Apache's HttpClient package was used to simulate the behaviour of a browser in order to retrieve results in real-time. Testing the system required constant internet connection without any interruptions by proxy server authentication programs. A modem connecting to an external network was used for this reason.

The integrated query interface was designed to include only the common attributes on airline sites. Some elements such as class and return trip were omitted because they are not common and do not affect the ability to retrieve results from the remote web server. An observation of similar online systems such as Travel Start showed that they also omit these fields. Complex fields were broken down as much as possible; for example, the date field comprises three parts which can then be combined in different ways to accommodate all such fields encountered.

Several schema matching approaches are described in the literature (Rahm and Bernstein 2001). The schema matching technique called pairwise-attribute correspondence (He and Chang 2007) was used to map semantically similar elements in different forms. This only considered schema information such as name, description, data type and structure but not instance data. Figure 2 shows query interfaces S1, S2, S3 and S4; and IS, the resulting integrated schema. Examples of synonymous attributes deduced from these different vocabularies through pairwise-attribute correspondence is shown with the IS.

By studying the query operators for each query interface, queries tailored to each query interface were built for the 4 airlines. In order to understand the query mechanisms, we analyzed the query interfaces' source pages. This meant learning the set of operators, and the rules to combine these operators with query parameters to create the query.

We located the form manually in the HTML source. If the form uses GET, name-value pairs will be listed in the URL address. If the form uses POST, name-value pairs are

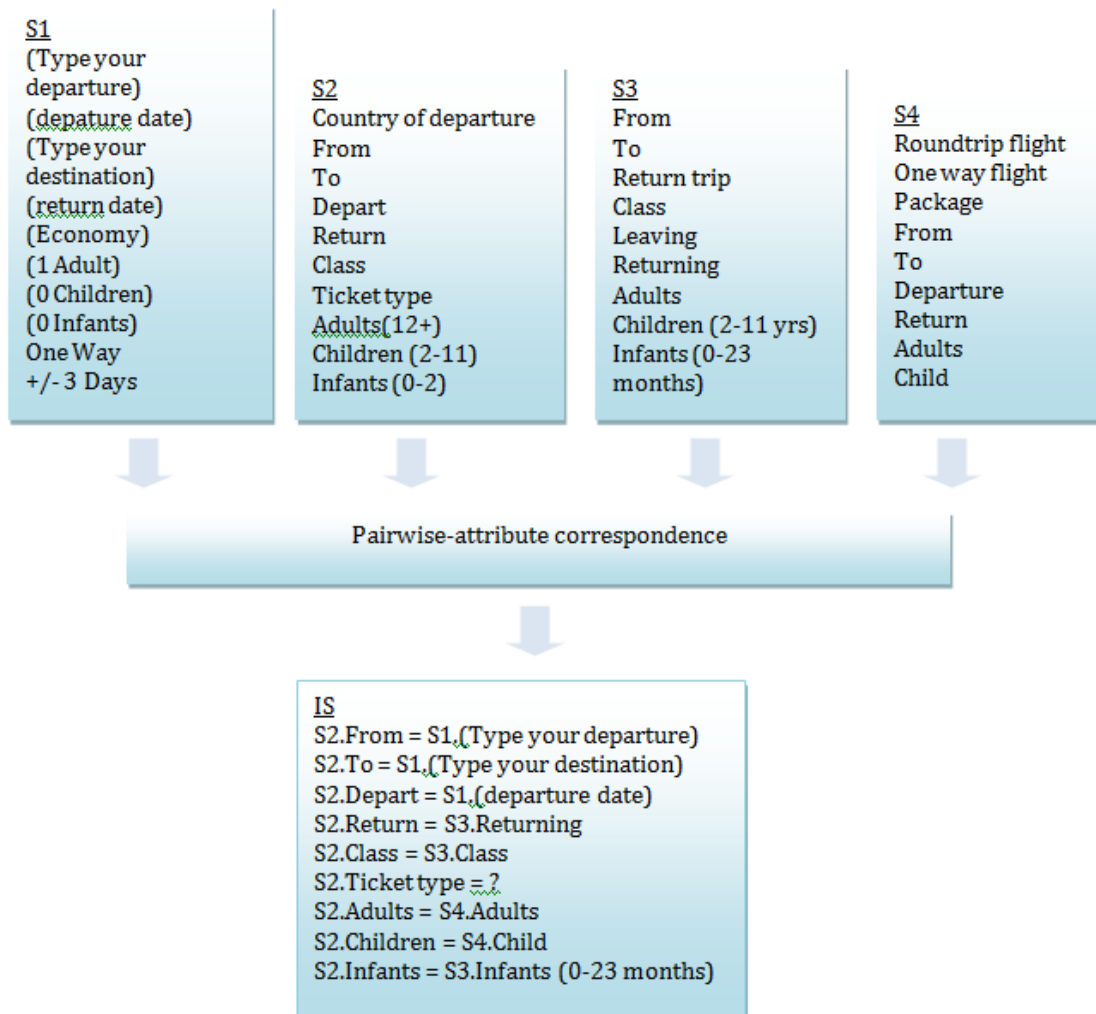


Figure 2: Example derivation of Integrated Schema (IS) and mappings to IS attributes.

embedded in the HTML page sent to the web server. Most sites reviewed use the HTTP POST method which is considered safer as data submitted to the remote server is not displayed in the address window and therefore cannot easily be modified. Mozilla Firefox's "View Page Info" right-click functionality recognizes web forms and lists the form fields; it is also able to pick up hidden form fields. As an example,

<http://booking.airnamibia.aero/checkrequirements.asp?origin=WDH&destination=CPT&departure=14&departuremonthyear=Oct 2008&returnday=20&returnmonthyear=Oct 2008&numberofadults=1&numberofchildren=0&numberofinfantswithoutseats=0&flightclass=GTKQHBMSY&mode=&faretype=lowest&sourcepage=requirements.asp>

is a typical URL query to retrieve flight details.

<http://booking.airnamibia.aero/checkingrequirements.asp> indicates the file on the web server from which results are obtained; the "?" separates the name-value pairs from the file; the "&" separates the name-value pairs.

In order to communicate with a remote web server, we used Apache's HttpClient package to submit queries. The client consists of GetMethod and PostMethod which are

similar to the GET and POST method recognized by server-side CGI scripts. PostMethod accepts a complete URL with name-value parameters specified.

A few problems were encountered when submitting URL queries programmatically, such as handling cookies using HttpClient. As an example, British Airways insists on setting the cookie policy before data can be retrieved from the web server. If the post does not return an HttpStatus.SC_OK this indicates the presence of errors or a redirection problem. A familiar occurrence with most of the query interfaces analyzed is redirection to another page when results are being generated. Once a redirection error is detected, the POST method reposts its name-value pairs to the redirection URL.

4. Result Processing

Result processing involves extracting relevant items from returned data, saving these correctly in the database, and then allowing the user to browse, sort and query the consolidated list. The bulk of this work is done by the Preparatory System, which stores in a database everything that the Runtime System needs to do its task. Saving flight results and allowing users to work with these is not hard. In contrast, data extraction requires three steps. These are: locating the data region on response pages; identifying blocks of data (single flight details) within this region; and removing noise and data duplication.

4.1 Extracting the data

Result pages are first passed to the HTML Editor Kit (HTMLEditor 2008) that corrects HTML by inserting missing tags, removing useless tags such as end-tags with no corresponding start-tag, and repairing end tags in the wrong order or illegal nesting of elements.

Airline pages returned by Hidden Web contain tables of results (with flight number, departure time, arrival time, class and prices). However, many pages contain several tables. Although different airline websites are designed by different people, these designers have a common consideration: as the data records are the focus, they are usually centered on the page. Unfortunately, the central area of result pages will often comprise tables nested inside other tables, and the parse trees generated can be complex. In all cases non-leaf nodes are tags and leaf nodes are string values, each localized between a pair of tags. Studies of these tags revealed that the data in the central table lies within `<TD>..</TD>` tags or else between `..` tags. The extractor method searches for tables, computes the type and number of each sub-region, and describes the structural layout of the region. From this it locates the main/central table and using the TD/span tags within this extracts each data item. Most of these items are common to all airlines and thus contribute to the final result returned by the integrated system. Other airline-specific items must be detected and discarded – we refer to these as noise.

By investigating pages returned by South African Airways, Kulula, Mango, British Airways, One Time and Kenya Airways, a common property was discovered. The length of noise is generally larger than that of pure data. The pure data like times, dates and flight numbers are generally similar in length even though their format varies. This helps in filtering required data from data noise. Thus the parser first restricts string length to

remove noise. As some noise has the same length as pure data, the parser then uses string pattern recognition to check remaining values like 12h30 or 5:50. Tests on many airlines showed that the length criterion successfully eliminated most noise e.g. “Unlimited free changes”, “R40 meal voucher” and “30 kg checked in luggage”.

4.2 Merging results

Net-Beans 6.0.1 has a powerful tool that automatically generates HTML pages. Its Do-Get and Do-Post methods are used to get data from result pages and subsequently to post consolidated results. The pure data from each airline are stored in a temporary table first, since values appear in different orders. From there they are transferred to the correct column of the final table, with duplicates removed. In the airlines domain, the main results – flight number, time, date and class – are returned directly, but airlines differ regarding pricing presentation. Some have a single value, others give component prices (e.g. flight cost, tax and total). The system detects such pricing and deduces which is the total cost. This value alone is the way that all prices are presented in the final merged results. The notation used for fields also differs e.g. time can be shown as 23:30 or 11h30, etc. The current system does not reformat these in a standard way, but this could easily be done since the semantics (i.e. that these are times) is known. Finally, there are other fields, like aircraft type for SAA, that are excluded from the consolidated results because they are not common. Since a link to each airline’s website is provided anyway (e.g. for booking flights), such additional site-specific information is reachable indirectly if needed.

5. Experimentation

The query processing and result interpretation components were evaluated separately on a number of hidden web sites. The system as a whole was also tested on the 4 airlines whose query forms had been learned by inspection. Tests investigated the following:

1. How successfully does the Query Preparation system interpret interfaces of airline databases on the Hidden Web?
2. How successfully does the Result Extraction system interpret responses from airline databases on the Hidden Web?
3. Does the integrated system retrieve the correct results from the pre-defined Hidden Web databases?

The effectiveness of the Query Preparation system was evaluated by surveying a dataset of 72 airline flight booking query interfaces. Four types of outcome were observed:

- A) The query form was successfully located and interpreted, and all relevant information correctly stored in the database;
- B) The correct query interface was partially parsed, that is, some of its form elements were correctly detected and interpreted;
- C) The wrong form interface was parsed and stored in the database, referred to as false positives;
- D) The query form was not extracted by the parser at all due to malformed HTML code.

A total of 39 query forms (54%) were correctly detected. However the semantic analysis of 11 of these was only partially correct – so only 28 pages, or 39% of the URLs visited,

were interpreted completely correctly. This low success rate can be attributed to complex web page design and layout; form-based query interfaces embedded with other forms on the same page; and scripting that generates the forms (e.g. SAA's query interface is generated by JavaScript). Of the remaining 33 sites, 5 resulted in false positives (type C) while as many as 28 could not be parsed due to malformed HTML (type D).

The result interpretation system was tested on pages returned by 55 airlines on the Hidden Web. The Extractor succeeded in fully interpreting 49 of the 55 pages, partially interpreted two, and completely failed to extract data from the remaining four. In the 49 cases fully extracted, the blocks of data were stored inside a Table region of the HTML page, and all relevant content was correctly detected and interpreted. In the 2 cases of partial extraction, some blocks of data were stored inside the Table region and successfully handled; but other blocks of data were retrieved from the hidden database by Javascript functions. Similarly, in the last 4 cases where extraction failed completely, all the data was produced by Javascript functions. Since 89% of randomly selected HTML result pages were fully extracted, we checked how well-known were the airlines inaccessible to our system. The four that completely failed to be interpreted were Zest-Airs, Aer Arann, Lot, and Air New Zealand; those partially extracted were Air Malta and Pakistan Airlines.

These first tests evaluated the extent to which the Query Preparation and Result Extraction subsystems were generic solutions by investigating what proportion of a random collection of Hidden Web airline databases they handled correctly. As we were unable to build a generic system for submitting queries, we could test the overall system only against 4 pre-defined airlines for which we wrote customized submission code. These airlines were: South African Airways (SAA), British Airways (BA), Air Namibia and Air Kenya. These airlines were chosen because their query interfaces were quick to learn.

In these tests queries were input to the integrated interface; the query subsystem sent corresponding queries to each of the 4 Hidden Web databases, and the result interpretation component extracted and merged the responses. These results were then validated by entering the same query on the individual forms provided by each of the 4 airlines and checking their responses against those of the system's consolidated list.

Only three of the four airlines were successfully queried. British Airways failed to produce results because of cookie restrictions. The BA web server is non-compliant with any of the cookie specifications supported by the HTTP client. In addition to correctness testing, we observed a marked difference in response time from different Hidden Web sites. Specifically, SAA took considerably longer than the others. This was partly due to larger result sets (more flights meeting query criteria) and hence more HTML pages being sent. Our prototype system accesses the databases sequentially; clearly a parallel solution is needed. Moreover the wide range of retrieval times indicates that such an approach also needs a thread that displays already-received portions of the consolidated result at the same time as other threads wait for the slower responses of the remaining databases.

6. Related work

A number of researchers are investigating integrated access to Hidden Web data, or aspects thereof. MetaQuerier (He et al 2005) also uses a preparatory system, with its form-filling database containing constraint templates for each form field which express query conditions on that field. HiWe (Raghavan and Garcia-Molina 2001) is another such system, which requires human assistance rather than being fully automated. Since HiWe is designed to learn from its human input, it should steadily improve as it learns more about a domain. Another system requiring human intervention is (Wu et al 2004), where a cluster-based field-matching approach is used to derive an integrated query form, and a cosine Information Retrieval function semi-automatically chooses field labels (Dragut, Yu and Meng 2006). Unlike our system which is designed specifically for the airline domain and hence aware of the fields to expect (times, flight numbers etc.) these are general systems designed to learn the fields of any given domain, and hence they employ heuristics to extract possible field labels and values, which makes them less accurate than our system designed from human inspection of airline forms.

The problem of query submission has been studied in e.g. DeepBot (Alvarez et al 2007) which overcomes limitations imposed by client-side scripting languages and session maintenance mechanisms by maintaining browser sessions containing all the required information after a URL is visited. Konopnicki and Shmueli (2005) propose building constraint templates that specify the format of an “acceptable query condition”. This specifies all the input fields that need to be filled-in in the form; along with metadata constraints in the HTML code, such as the length of an input field or type. Wu, Doan & Yu (2006) use extraction patterns consisting of a cue phrase and completion; where a completion is a list of noun phrases considered to be possible instances. Liddle, Yau and Embley (2002) compare the HTTP GET and POST methods and note that GET ignores important metadata that might suggest possible domains for the field or input constraints such as limited field length.

Liddle et al (2002) studied result display, such as handling error messages and sites that return a few results per page (with a link to the next page). They note that error pages can be detected before parsing because their size tends to be constant and much smaller than actual results; but they need to be distinguished from “no result” cases. Data integration across different sites has been studied both practically (Halevy 2001, He and Chang 2006) and theoretically (Lenzirini 2002), with schema mapping presenting the biggest challenge. Once again, this is because these systems are designed to handle any domain, rather than being customized for just one such as airlines. Where they do focus on one domain e.g. Travel Start, they are not generic, accessing only a pre-determined subset of Hidden databases in that domain.

7. Conclusion

We proposed a domain-specific Hidden Web query system and built a prototype that successfully queries multiple pre-determined Hidden Web databases. We studied the domain of airlines and, based on this, designed generic components able to parse correctly a reasonable proportion (39%) of random airline query interfaces. Our generic result extractor automatically interpreted the vast majority (89%) of responses from airline databases on the Web. However, we did not manage to automatically submit queries to arbitrary airlines, because the time taken to learn query interfaces was

greater than expected, with the result that a generic query submission system remains as future work.

Some additional development problems encountered included poor Internet performance slowing down testing, and UCT's authentication proxy not allowing external connections programmatically via Java thus requiring an external network to be used instead. Overall we believe that domain-specific software for integrated query of the Hidden Web is the most promising way of replacing semi-automated solutions currently in existence with a fully-automated alternative.

References

- Alvarez, M., Raposo, J., Pan, A., Cacheda, F., Bellas, F. and Carneiro, V. 2007. DeepBot: A Focused Crawler for Accessing Hidden Web Content. In: *Proceedings of the 3rd International Workshop on Data Engineering Issues in E-commerce and Services*, June 12, 2007: 18-25.
- Bergman, M. 2001. The deep web: surfacing hidden value. *Journal of Electronic Publishing*, 7(1):3-21.
- Dragut, E. C., Yu, C. and Meng, W. 2006. Meaningful Labelling of Integrated Query Interfaces. In: *Proceedings of the 32nd international conference on Very Large Data Bases*, 2006. Seoul:679-690.
- Halevy, Y. A., 2001. Answering queries using views: A survey. In: *Proceedings of the 27th International Conference on Very Large Data Bases*, December 2001. Springer: New York:270–294.
- He, B., Patel, M., Zhang, Z. and Chang, K. C. 2005. Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web. In: *Proceedings of the 2nd CIDR Conference*, 2005.
- He, B. and Chang, K. C. 2006. Automatic Complex Schema Matching Across Web Query Interfaces: A Correlation Mining Approach, March 2006. New York:346-395.
- He, B. and Chang, K. C. 2007. Statistical Schema Matching across Web Query Interfaces. In: *Proceedings of the ACM International Conference on Management of Data, 2007: 217-228*.
- HTMLEditor Kit. Available: <http://java.sun.com/j2se/1.4.2/docs/api/javawebkit/html/HTMLEditorKit.html> (accessed 15 September 2008)
- Jericho HTML Parser. Available: <http://jerichohtml.sourceforge.net/doc/index.html> (accessed 15 September 2008).
- Konopnicki, D. and Shmueli, O. 2005. Database-Inspired Search. In: *Proceedings of the 31st International Conference on Very Large Data Bases*, 2005. Trondheim:2-12.
- M. Lenzerini. 2002. Data Integration: A theoretical Perspective. In: *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, June 2002. Rome:233-246.
- Liddle, S.W., Embley, D.W., Scott, D.T. and Yau, S.H. 2002. Extracting Data behind Web Forms. In: *Proceedings of the 28th International Conference on Very Large Databases*, 2002. Hong Kong:2-11.
- Liddle, S. W., Yau, S. H. and Embley, D. W. (n.d.). 2002. Extracting Data behind Web Forms. Brigham Young University, 2002.
- Raghavan, S. and Garcia-Molina, H. 2001. Crawling the Hidden Web. In: *Proceedings of the 27th International Conference on Very Large Databases*, 2001. Rome:129-138.

Rahm, E. and Bernstein, P.A. 2001. A survey of approaches to automatic schema matching. *VLDB Journal* 10(4):334-350.

Travel Start Website Result page. Available: <http://www.travelstart.co.za/> (accessed 8 November 2008).

Wu, W. and Doan, A. 2006. WebIQ: Learning from the Web to Match-Deep-Web Query Interfaces. In: *Proceedings of the 22nd International Conference on Data Engineering*, April, 2006.

Wu, W., Yu, C., Doan, A. and Meng, W. 2004. An interactive Clustering-based Approach to Integrating Source Query Interfaces on the Deep Web. In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, June, 2004.

END