An XML-based Wrapper Generator for Web Information Extraction

Ling Liu, Wei Han, David Buttler, Calton Pu, Wei Tang

Oregon Graduate Institute of Science and Technology
Department of Computer Science and Engineering
P.O.Box 91000 Portland, Oregon 97291-1000 USA
{lingliu,weihan,buttler,calton,wtang}@cse.ogi.edu

1 Introduction

There has been tremendous interest in information integration systems that automatically gather, manipulate, and integrate data from multiple information sources on a user's behalf. Unfortunately, web sites are primarily designed for human browsing rather than for use by a computer program. Mechanically extracting their content is in general a rather difficult job if not impossible [4]. Software systems using such web information sources typically use hand-coded *wrappers* to extract information content of interest from web sources and translate query responses to a more structured format (e.g., relational form) before unifying them into an integrated answer to a user's query. The most recent generation of information mediator systems (e.g., Ariadne [3], CQ [5, 7], Internet Softbots [4], TSIMMIS [2]) addresses this problem by enabling a pre-wrapped set of web sources to be accessed via database-like queries.

However, hand-coding a wrapper is time consuming and error-prone. We have also observed that, by using a good design methodology, only a relatively small part of the code deals with the source-specific access details, the rest of the code is either common among wrappers or can be expressed in a high level, more structured fashion. As the Web grows, maintaining a reasonable number of wrappers becomes impractical. First, the number of information sources of interest to a user query can be quite large, even within a particular domain. Second, new information sources are constantly added on the Web. Thirdly, the content and presentation format of the existing information sources may change frequently and autonomously.

With these observations in mind, we have developed a wrapper generation system, called XWrap, for semi-automatic construction of wrappers for Web information sources. The system contains a library of commonly used functions, such as receiving queries from applications, handling of filter queries, and packaging results. It also contains some source-specific facilities that are in charge of mapping a mediator query to a remote connection call to fetch the relevant pages and translating the retrieved page(s) into a more structured format (such as XML documents or relational tables). A distinct feature of our wrapper generator is its ability to provide an XML-enabled, feedback-based, interactive wrapper construction facility for Internet information sources. By XML-enabled we mean that the extraction of information content from the Web pages will be captured in XML form and the process of filter queries is performed against XML documents. By feedback-based we mean that the wrapper construction process will be revisited and tuned according to the feedback received by the wrapper manager.

2 Methodology

The philosophy behind our "XML-enabled" wrapper generation methodology is to develop mechanisms that provides a clean separation of the semantic knowledge of information extraction from the wrapper code generation using a rule-based approach. More concretely, the wrapper generator first exploits formatting information in Web pages to hypothesize the underlying semantic structure of a page, and then encode the hypothetical structure and the information extraction knowledge of the web pages in a rule-based declarative language designed specifically for XWrap information extraction. From the set of information extraction rules and the XML-templates derived throughout the XWrap walkthrough sessions, the system constructs a wrapper program that facilitates both the tasks of querying of a semi-structured Web source and integrating it with other Web information sources. We partition the wrapper generation process into a series of sub-processes called *phases*, as shown in Figure 1. A phase is a logically cohesive operation that takes as input one representation of the source document and produces as output another representation.

The first phase is called the *structure analysis*. It performs the encoding of information extraction knowledge into a set of implementation-independent information extraction rules. It involves three tasks: (1) fetching a Web page from a remote site and generating a tree-like structure for the page, (2) identifying regions and tokens that are key components or key element templates for extracting information content from the page, and (3) inferring the nesting hierarchy of sections, which represents the syntactic structure of the page. For a Web page in HTML, this phase builds a source-specific HTML parser that wraps the page into a tree structure, annotates content tokens in comma-delimited format and nesting hierarchy in an XML-template file (see Section 3 for an example).

The second phase is called the *source-specific XML generation*. It involves two steps: (1) Generate a source-specific XML template file based on the content tokens and the nesting hierarchy specification obtained in the structure analysis

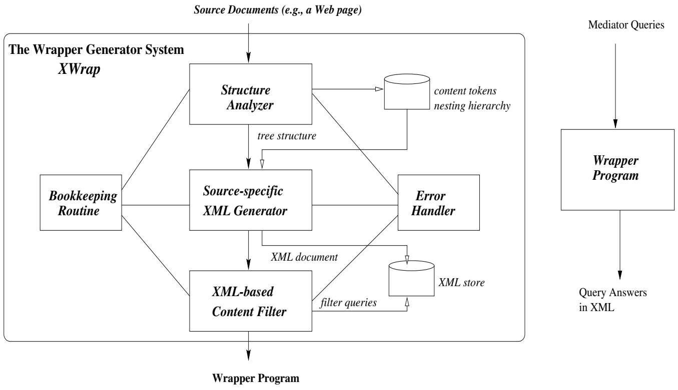

Figure 1: Phases of an XML-enabled Wrapper Generator

phase; and (2) Construct a source-specific XML generator using the template-based XML generator. An XML template is a well-formed XML document. In addition to constructs that one would normally expect in an XML file, it also contains a small set of processing instructions and special placeholders defined by the XML generator.

The third phase is called the *generic XML-based content filter (content extraction controller).* After wrapping a Web source in an XML format, we build a generic XML-based content filter that is capable of handling complex conjunctive or disjunctive queries handed over by various mediator applications.

The *bookkeeping* routine of the wrapper collects information about all the data objects that appear in the source document, keeps track of the names used by the program, and records essential information about each. For example, a wrapper needs to know how many arguments a tag expects, whether an element represents a string or an integer.

The *error handler* is designed for the detection and reporting errors in the source document. It is invoked when a flaw in the source document is detected. It must warn the wrapper developer by issuing a diagnostic, and adjust the information being passed from phase to phase so that each phase can proceed. It is desirable that wrapper construction be completed on flawed source documents, at least through the structure-analysis phase, so that as many errors as possible can be detected in one construction pass. The error messages should allow the wrapper developer to determine exactly where the errors have occurred. Both the bookkeeping and error handling routines interact with all phases of the wrapper generator.

3 A Walk Through Example

3.1 Information Extraction

Consider the weather report page for Savannah, GA at the national weather service site, and a fragment of HTML document for this page in Figure 2.

Figure 3 shows a portion of the HTML tree structure, corresponding to the above HTML fragment, which is generated by running a source-specific parser on the Savannah weather source page. In this portion of the HTML tree, we have the following six types of tag nodes: `TABLE`, `TR`, `TD`, `B`, `H3`, `FONT`, and a number of semantic token nodes at leaf node level, such as `Maximum Tempature`, `Minimum Tempature`, `84.9(29.4)`, `64.0(17.8)`, etc.

In the current implementation, the structure analyzer sets up three panels in an information extraction window. The two top level panels displays the web page and its HTML tree structure, and the bottom panel is used to display the information extraction rules and the comma delimited file generated.

Based on the interaction with the wrapper developer, the region extractor identifies four semantic regions of interest for extraction, each is a table in the nws.noaa.gov web page. The semantic-token extractor infers that the leaf node `Maximum and Minimum Temperatures` is the heading of a table section, the string `Maximum Tempature F(C)` is a semantic token with the string itself as the token name and the string `84.9 (29.4)` as the token value, and so forth. By traversing the entire tree structure, the semantic-token extractor produces as output, a set of information extraction rules for extracting the semantic tokens into a comma-delimited file and the hierarchical structure into an XML-tamplate file for each of the nws.noaa.gov current weather report page. Consider the fragment of the tree structure given in Figure 3. By walking through the left most branch of the tree with the user's feedback, XWrap can infer that `Maximum and Minimum Temperatures` is the table name because it is in between a pair of header tags `<H3>` and `</H3>`. Also based on the observation that all the rest of children nodes of `TABLE` are of the same type `TR`, and each again has three children nodes. The leaf node `Maximum and Minimum Temperatures` of the first branch of `TABLE` is marked as the table name, the leaf nodes `Maximum Temperature`, `Minimum Temperature` of the second branch of `TABLE` as column names, and each of the rest of the branches of `TABLE` as an instance of the second `TR` node type. We develop an algorithm that, given a page with all regions (tables and text sections, and headings identified, outputs an XML template for the page. Figure 4 shows the fragment of the XML template file corresponding to the part of a NWS weather report page shown in Figure 3.

Due to the fact that the heuristics used for identifying sections and headings may have exceptions for some information sources, it is possible for the system to make mistakes when trying to identify the hierarchical structure of a new page. For example, based on the heuristic on font size, the system may identify some words or phrases as headings when they are not, or fail to identify phrases that are headings, but do not conform to any of the pre-defined regular expressions. We have provided a facility for the user to interactively correct the system's guesses. Through a graphical interface the user can highlight tokens that the system misses, or delete tokens that the system chooses erroneously. Similarly, the user can correct errors in the system-generated XML-template that describes the structure of the page.

3.2 Source-specific XML generator

The source-specific XML generator consists of an XML Template Engine and an XML parser. The XML template engine generates XML statements using both the comma-delimited file and the XML template parsed by the XML parser. Comparing with the normal XML documents, XML templates are well-formed XML files that contain processing instructions. Such instructions are used to direct the template engine to the special placeholders where data fields should be inserted into the template. For instance, the processing instruction **XG-InsertField-XG** has the canonical form of

```
<TABLE><TR><TD COLSPAN=3><H3><FONT FACE="Arial, Helvetica">Maximum and Minimum Temperatures</FONT>
</H3> </TD></TR><TR><TD ALIGN=CENTER BGCOLOR="#FFFFFF"><B><FONT COLOR="#0000A0"><FONT FACE=
"Arial,Helvetica">Maximum<BR>Temperature<BR>F(C)</FONT></FONT></B></TD><TD ALIGN=CENTER BGCOLOR=
"#FFFFFF"><B><FONT COLOR="#0000A0"><FONT FACE="Arial,Helvetica">Minimum<BR>Temperature<BR>F(C)
</FONT></FONT></B></TD><TD></TD></TR><TR><TD ALIGN=CENTER><FONT FACE="Arial,Helvetica">82.0(27.8)
</FONT></TD><TD ALIGN=CENTER><FONT FACE="Arial, Helvetica">62.1(16.7)</FONT></TD><TD><FONT FACE=
"Arial, Helvetica">In the <B>6 hours</B> preceding Oct 29, 1998 - 06:53 PM EST / 1998.10.29 2353
UTC</FONT></TD></TR><TR><TD ALIGN=CENTER><FONT FACE="Arial,Helvetica">80.1(26.7)</FONT></TD>
<TD ALIGN=CENTER><FONT FACE="Arial, Helvetica">45.0(7.2)</FONT></TD><TD><FONT FACE="Arial,
Helvetica">In the <B>24 hours</B> preceding Oct 28, 1998 - 11:53 PM EST / 1998.10.28 0453 UTC</FONT>
</TD></TR><TR><TD COLSPAN=3><HR SIZE=1 NOSHADE WIDTH="100%"></TD></TR></TABLE> ......
```

Figure 2: An HTML fragment of the weather report page at nws.noaa.gov site

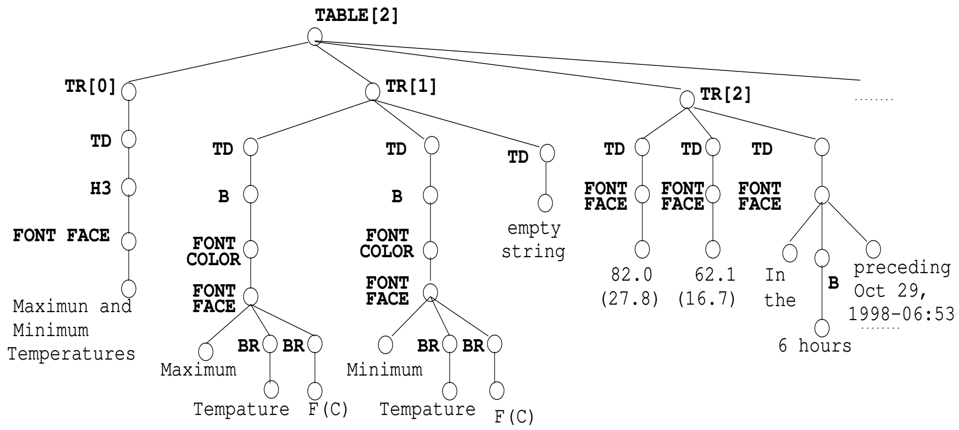

Figure 3: Another fragment of the HTML tree for the Savannah weather report page

<?XG-InsertField-XG ‘‘FieldName"?>. It looks for a field with a specified name "FieldName" in the comma-delimited file and inserts that data at the point of the processing instruction.

An XML template also contains a repetitive part. The XG-Iteration-XG processing instruction determines the beginning and the end of a repetitive part. A repetition can be seen as a loop in classical programming languages. After the template engine reaches the "End" position in a repetition, it takes a new record from the delimited file and goes back to the "Start" position to create the same set of XML tags as in the previous pass. New data is inserted into the resulting XML file.

3.3 XML-based Content Filter

The XML-based Content Filter is the interface between the mediator application and the wrapper. The main tasks of the content filter include

- Accept a mediator query, identify network locations of the pages needed to answer the query, and call the data wrapper manager to wrap the source page(s) into the XML format;
- Translate the complex content-sensitive mediator queries over the given Web pages to SQL-like XML queries.

For single-document sources, this is straightforward, i.e., the URL for that page is known to the wrapper. For sources with multiple documents of the same type, a mapping between a query and the URL of the relevant page may be required. To provide the capability of determining the network location of the page relevant to a query, the wrapper developer specifies a mapping function which takes necessary arguments from a query (e.g., the city name and its four-character code on the NWS weather source) and constructs a URL pointing to the page to be fetched. For instance, in the NWS weather report site there is a one to one mapping between the city name and the URL of the page for that city. This mapping can be obtained by querying the city from either the US weather report home page or the world weather report home page. Currently we use Java programs for the purpose of making HTTP connections to the Web information sources and retrieving data from them.

To support information filtering over XML documents, we would need to understand the data structure used to hold the XML documents and the types of query operators a system would like to support. One popular approach is to use XML-QL style of query languages [1]. The main problems with this proposal is the availability of the software to support such an XML-QL based system. Another approach is to transform an XML document into a collection of relational tables or a class of complex objects. There are two main challenges in transforming an XML document to a collection of relational tables (or object classes). First, given an XML document, we need to decide how many relational tables are needed to capture the nesting structure of the XML file. Second, for multi-document sources, the XML

```
......
<Maximum_and_Minumum_Temperatures>
<Description>Maximum and Minimum Temperatures</Description>
<!-- Start of the repetition -->
<?XG-Iteration-XG ‘‘Start"?>
  <Maximum_and_Minimum_Temperatures_Child>
    <Maximum_Temperature>
    <Description>MaximumTemperature F(C)</Description>
    <Value><?XG-InsertField-XG ‘‘Maximum Tempature"></Value>
    </Maximum_Temperature>

    <Minimum_Temperature>
    <Description>MinimumTemperature F(C)</Description>
    <Value><?XG-InsertField-XG ‘‘Minimum Temperature"></Value>
    </Minimum_Temperature>

    <TD>
    <Description></Description>
    <Value><?XG-InsertField-XG ‘‘TD"></Value>
    </TD
  </Maximum_and_Minumum_Temperatures_Child>
<?XG-Iteration-XG ‘‘End"?>
<!-- End of the repetition -->
</Maximum_and_Minumum_Temperatures>
......
```

Figure 4: An Example XML template for a portion of the NWS current weather report page

document generated at Phase two is only an instance of the corresponding source. Quite often it may have some missing sections, or missing columns of a table section. Thus, the relational tables that capture the given XML document may not be generic enough to capture other instance pages of the same source. For example, consider the NWS current weather report source. For some cities in US, the weather report at a given time point may include the `Precipitation Accumulation` section, although at other times this section is not available. Another example is the case when some column or sub-section (such as the `Dew Point` field) is missing in one document instance but appeared in another instance document. Therefore, instead of using a pre-defined number of relational tables and pre-defined table formats, the design of the content filter should determine the number of relational tables to use and the table formats at run-time.

4 Description of Demo

We demonstrate the latest version of our wrapper toolkit as described in the previous sections. Specifically we show how to use our wrapper generator toolkit to construct wrappers for the following three classes of Internet information sources.

1. Web sources that are multiple-instance sources (i.e., multiple documents of the same type), and organize their content information in multiple two-dimensional tables, such as the current weather report source at the national weather service site, the stock quote information source at stockmaster.com or Yahoo investment site.

2. Web sources that are multiple-instance sources, and organize the content information in multiple nesting sections, such as the CIA factbook web site or the Sun Site Web Museum site.

3. Web sources that are single-instance sources, such as SIGMOD'99, ICDE'99 or VLDB'99 conference web sites.

Although all three classes of Web sources support different search and access methods, the wrappers generated hide all source specific details from the applications and end-users by providing a common interface to the underlying data independently of where and how it is stored. Furthermore, we demonstrate the filter query processing capabilities of our wrappers.

As part of the Continual Queries project, we have also developed a graphical browsing tool that lets users submit queries to wrappers, navigate through the XML files generated for the Web pages that have been wrapped, and zoom in on the CQ answer objects and their nested hierarchies as necessary.

Acknowledgement

This research is partially supported by DARPA contract MDA972-97-1-0016, Intel, and Boeing. Our thanks are also due to the Master students involved in the CQ and XWrap projects at OGI, especially Divya Kumar, Shujing Liu, Jaya Shrivastav, and Iffath Zofishan.

References

[1] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. *http://www.w3c.org/TR/1998/NOTE-xml-ql-19980819*, 1998.

[2] J. Hammer, M. Brennig, H. Garcia-Molina, S. Nesterov, V. Vassalos, and R. Yerneni. Template-based wrappers in the tsimmis system. In *Proceedings of ACM SIGMOD Conference*, 1997.

[3] C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, P. J. Modi, I. Muslea, A. Philpot, and S. Tejada. Modeling web sources for information integration. In *Proceedings of AAAI Conference*, 1998.

[4] N. Kushmerick, D. Weil, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of Int. Joint Conference on Artifical Intelligence (IJCAI)*, 1997.

[5] L. Liu, C. Pu, and W. Tang. Continual queries for internet-scale event-driven information delivery. *IEEE Knowledge and Data Engineering*, 1999. Special Issue on Web Technology.

[6] L. Liu, C. Pu, W. Tang, and W. Han. Conquer: A Continual Query System for Update Monitoring on the WWW. *International Journal of Computer Systems, Science and Engineering*, 1999. Special Issue on WWW Semantics, edited by Dan Suciu and Letizia Tanca.

[7] L. Liu, C. Pu, W. Tang, J. Biggs, D. Buttler, W. Han, P. Benninghoff, and Fenghua. CQ: A Personalized Update Monitoring Toolkit. In *Proceedings of ACM SIGMOD Conference*, 1998.