Interactive Web-Wrapper Construction for Extracting Relational Information from Web documents

Tsuyoshi Sugibuchi
Hokkaido University, Meme Media Laboratory
Kita 13 Nishi 8, Kita-ku, Sapporo,
Hokkaido Japan
buchi@meme.hokudai.ac.jp

Yuzuru Tanaka
Hokkaido University, Meme Media Laboratory
Kita 13 Nishi 8, Kita-ku, Sapporo,
Hokkaido Japan
tanaka@meme.hokudai.ac.jp

ABSTRACT

In this paper, we propose a new user interface to interactively specify Web wrappers to extract relational information from Web documents. In this study, we focused on improving user's trial-and-error repetitions for constructing a wrapper. Our approach is a combination of a light-weight wrapper construction method and the *dynamic previewing interface* which quickly previews how generated wrapper works. We adopted a simple algorithm which can construct a Web wrapper from given extraction examples in less than 100 milliseconds. By using the algorithm, our system dynamically generates a new wrapper from a stream of user's mouse events for specifying extraction examples, and immediately updates a preview result that shows how the generated wrapper extracts HTML nodes from a source Web document. Through this animated display, a user can make a lot of wrapper construction trials with various different combinations of extraction examples by only moving a mouse on the Web document, and reach a good set of examples to obtain an intended wrapper in a short time.

Categories and Subject Descriptors
H5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces (GUI)*

General Terms: Algorithms, Design, Human Factors

Keywords: User interfaces, Information extraction, Web wrappers

1. INTRODUCTION

A Web wrapper is used to convert an HTML document into a form that intended software can understand. A Web wrapper is a program which extracts data of interest from an HTML document and outputs the data in a structured form. Various wrapper construction tools [1] have been proposed for users to easily construct new Web wrappers for arbitrary Web documents. Wrapper construction from given extraction examples is one of the most popular approaches for such tools. In this approach, a user gives a set of portions of an HTML document as examples. Then the system constructs a new wrapper that extracts portions similar to given examples from the HTML document. The extraction manner of the wrapper depends on the given examples. Therefore, a user needs to make many trials with various combinations of examples until an intended wrapper is generated. Various interfaces for Web wrapper construction are proposed in previous studies. However, there are few studies that focus on the improving of this trial-and-error task.

In this study, we tried to improve this task through user's interactive manipulation and dynamic visual feedback from a system. Our *dynamic previewing interface* allows users to quickly change a combination of examples and to get an immediate visual feedback about an effect of the change. By using this interface, a user can dynamically make a lot of wrapper construction trials and easily find a desirable one.
To realize the dynamic previewing interface, we developed a data extraction model and a wrapper construction method for our Web wrapper. Our method constructs a wrapper without traversing the whole of an HTML document. Therefore it can construct wrappers in a significantly short time to realize the dynamic preview display.

2. DYNAMIC PREVIEWING INTERFACE

Our dynamic previewing interface for wrapper construction is a combination of user's interaction and visual feedbacks. In our dynamic previewing interface, an extraction example is selected by a mouse. Along the movement of the mouse pointer, the system dynamically changes a combination of extraction examples and generates a new wrapper immediately. Then the system updates its display highlighting extracted portions. A response time of this visual feedback is less than 100 milliseconds, so that users feel as if the preview display is animated along with user's mouse operations. In our system, users can make a lot of trials with various combinations of examples only through mouse movement, and find a good combination through the animated preview display.

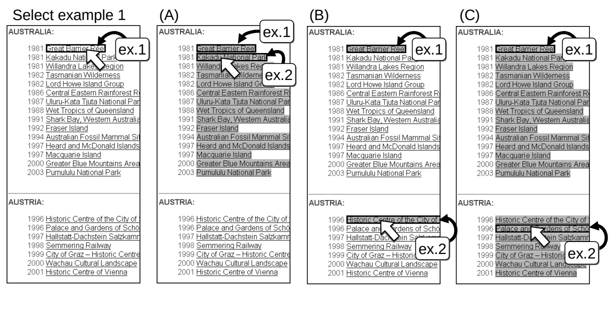

Figure 1. Example of the dynamic previewing display

Figure 1 shows that our interface smoothly highlights nodes to be extracted in response to the mouse movement. In this example, a user is specifying a Web wrapper on 'The World Heritage List' page presented by UNESCO. He selects the first candidate to extract on the page, and then he moves a mouse pointer over the page to specify various portions to work as the second candidate to extract. The specification of a new portion as the second candidate dynamically highlights all the portions to be extracted by generalizing the first two example candidates. The dynamically updated preview suggests various patterns of extraction results such as (A) all heritages of one nation, (B) all the heritages that appear in the first portion in some nation's heritage list, and (C) all the heritages of all the nations. User can quickly find good extractions by only moving the mouse pointer over the target Web page.

3. DATA EXTRACTION MODEL

Our Web wrapper extracts a set of HTML nodes from a source HTML document, and then the wrapper constructs a relational table from extracted nodes by associating related nodes as a tuple.

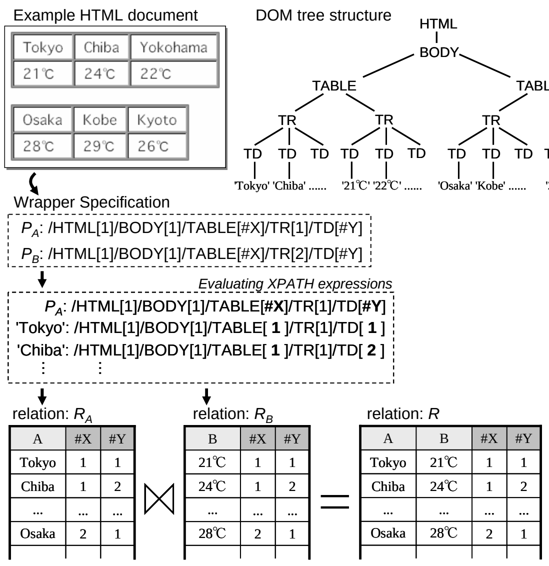

Figure 2. Data extraction model

A specification of our Web wrapper is represented as a set of XPATH expressions with variables. In this paper, a variable in an XPATH expression is denoted by an alphabet prefixed with '*#*' like '*#X*'. When an XPATH expression is evaluated to extract HTML nodes, a variable in the expression works as a wild card which matches for any type of a node or any position. Then a node type or a position corresponding to the variable is output as a value of the variable. When the wrapper constructs a relation from extracted HTML nodes, the wrapper construct tuples by comparing values of variables. Figure 2 shows an example of our data extraction process. In this example, XPATH expressions P_A and P_B extract HTML nodes as values of attributes *A* and *B* in the relation *R*. Both XPATH expressions include variables *#X* and *#Y*. As a result, the extracted result for P_A forms a relation R_A that consists of extracted node *A*, and values of *#X* and *#Y*. In the same way, R_B is the extracted result for P_B. Then our wrapper constructs relation *R* as the natural join of R_A and R_B.
Our data extraction method works well if HTML nodes we intend to extract appear in a repetitive structure in the DOM tree. The method can be applied not only to basic HTML tables, but also to various hierarchical structures and transposed tables that are difficult for conventional delimiter-based wrappers to extract data from.

4. WRAPPER CONSTRUCTION METHOD

For end users to construct a Web wrapper by end users, we define two types of user's actions we call *generalization* and *aggregation*. In both actions, a user selects two HTML nodes from an HTML document. By generalization, selected two nodes are treated as extraction examples in the same attribute of a relation. By aggregation, selected two nodes are treated as extraction examples in the same tuple of a relation.
Figure 3 shows an example of user's actions. First, the user specifies a generalization on two cells *a1* and *a2*. By this action, all cells in the first row of each table are extracted as an attribute A of the relation R. Then the user specifies an aggregation on the two cells *a2* and *b1*. By this action, all the cells in the second row of each table are extracted as a new attribute B, and then each cell in the first low, the one containing a city name, and the cell just under it, the one containing the temperature of the city, are combined into a single tuple.

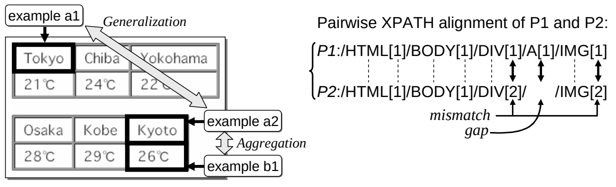

Figure 3. User's actions and an XPATH alignment

In out wrapper construction method, we define a set of XPATH expressions from a sequence of the user's actions. In each action, we define two XPATH expressions that point two selected nodes, and compare the two expressions to assign variables to XPATH expressions. To compare two XPATH expressions, our system computes the *XPATH alignment*. Figure 3 shows an example of the XPATH alignment. The XPATH alignment is a mutual arrangement of location steps in two XPATH expressions. To obtain the XPATH alignment, we use an algorithm for computing string alignments through dynamic programming [2].

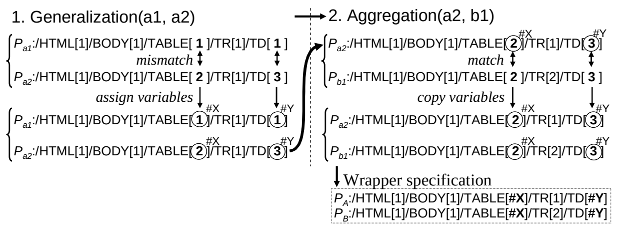
1. Generalization(a1, a2)
2. Aggregation(a2, b1)
P_{a1}:/HTML[1]/BODY[1]/TABLE[1]/TR[1]/TD[1]
mismatch
P_{a2}:/HTML[1]/BODY[1]/TABLE[2]/TR[1]/TD[3]
assign variables
P_{a1}:/HTML[1]/BODY[1]/TABLE[①]/TR[1]/TD[①]
P_{a2}:/HTML[1]/BODY[1]/TABLE[②]/TR[1]/TD[③]
#X
#Y
P_{a2}:/HTML[1]/BODY[1]/TABLE[②]/TR[1]/TD[③]
match
P_{b1}:/HTML[1]/BODY[1]/TABLE[2]/TR[2]/TD[3]
copy variables
P_{a2}:/HTML[1]/BODY[1]/TABLE[②]/TR[1]/TD[③]
P_{b1}:/HTML[1]/BODY[1]/TABLE[②]/TR[2]/TD[③]
Wrapper specification
P_A:/HTML[1]/BODY[1]/TABLE[#X]/TR[1]/TD[#Y]
P_B:/HTML[1]/BODY[1]/TABLE[#X]/TR[2]/TD[#Y]

Figure 4. Wrapper construction process

For each generalization action, we assign new variables for each mismatched pair of node tests or predicates in the XPATH alignment. In this example, variables *#X* and *#Y* are assigned to predicates after TABLE and TD in P_{a1} and P_{a2}. For each aggregation action, we equalize arrangements of variables in each matched pair of node tests and predicates. In this example, variables *#X* and *#Y* are copied from P_{a2} into P_{b1}.

5. CONCLUDING REMARKS

In this paper, we proposed a new user interface to construct a Web wrapper. Immediate visual feedback of our interface allows users to easily explore in a large set of wrapper candidates. In our future work, we would like to extend proposed technique to construct a Web wrapper for multiple Web pages, and apply this interface to an information integration environment that allows users to interactively combine various Web resources.

6. REFERENCES

[1] Alberto H. F. Laender, et al. A brief survey of web data extraction tools. ACM SIGMOD Record, 31(2), pp. 84-93, 2002.
[2] Dan Gusfield. Algorithms on Strings, Trees, and Sequences. Cambridge Press, pp. 213-223, 199