# Schema-Guided Wrapper Maintenance for Web-Data Extraction

Xiaofeng Meng, Dongdong Hu, Haiyan Wang
School of Information,
Renmin University of China,
Beijing 100872, China
xfmeng@mail.ruc.edu.cn

Chen Li
School of Information and CS
University of California, Irvine,
CA 92697-3425,USA
chenli@ics.uci.edu

## Abstract

Extracting data from Web pages using wrappers is a fundamental problem arising in a large variety of applications of vast practical interest. There are two main issues relevant to Web-data extraction, namely wrapper generation and wrapper maintenance. In this paper, based on a prototype system, called *SG-WRAP*, which can effectively generate wrappers (extraction rules) to extract data from given HTML pages. We propose a novel schema-guided approach to automatic wrapper maintenance. It is based on the observation that despite various page changes, many important features of the pages are preserved, such as syntactic features, annotations, and hyperlinks. Our approach uses these preserved features to identify the locations of the desired values in the changed pages, and repair wrappers correspondingly. Our intensive experiments over 16 real-world Web sites show that the proposed automatic approach can effectively maintain wrappers to extract desired data with high accuracies.

## 1 Introduction

The World Wide Web has become one of the most important connections to various information sources. A large proportion of the Web data is embedded in HTML documents. This language serves the visual presentation of data in browsers, but not for automated, computer-assisted information management systems. Thus if data from different sources needs to be integrated, it is necessary to develop special and often complex programs to extract data from Web pages. To achieve this goal, people have developed w*rappers* [7], which are specialised programs that can automatically extract data from Web pages and convert the information into a structured format. Different methods [1, 2, 5, 7, 8, 9, 13, 14, 15, 17] have been proposed to automate the wrapper-generation process.

There are many challenges in constructing wrappers. Often, a wrapper needs to be developed for each data source because of the heterogeneous page structures from different web sites. Thus generating wrappers for different sources could be time consuming and error prone. In addition, Web pages are extremely dynamic and continually evolving, which results in frequent changes in their structures. Consequently, wrappers may stop working in the presence of these changes. It is often critical to update or even completely rewrite existing wrappers, so that we can still extract the desired data. One way to maintain wrappers is to re-create wrappers from scratch using the new pages. But this method is inefficient due to the heavy workload to the system developers.

Recently, several methods are presented to address the problem of automatically repairing (maintaining) web wrappers. Kushmerick [11, 15] define a sub-problem of it, called *wrapper verification*, which checks if a wrapper stops extracting correct data. Their proposed solution analyzes pages and extracted information, and detects the page changes. If she finds that the pages have changed, the designer is notified; then she can relearn the wrapper from the pages with the new structure. Knoblock at el. [8] developed a method for wrapper repairing in the case of small mark-up changes. Chidlovskii [4] presents an automatic-maintenance approach, which can repair wrappers under the assumption that there are only small changes.

In this paper, we propose a novel schema-guided approach to wrapper maintenance, which is based on our previous work of schema-guided wrapper generator SG-WRAP[14,15]. The maintenance solution is based on the following observations. Although changes of HTML documents are various, some features of desired information in pages are often preserved, such as syntactic features of data items, possible hyperlink, and annotations (see section 3.1). In addition, user often express the same targets to extract, so the underlying schemas for the extracted data often do not change. It is feasible to recognize data items in the changed pages using these features. We maintain wrappers in four steps. At First, features are obtained from the user-defined schema, previous extraction rule, and the extracted results. Secondly, we recognize the data items in the changed page with these features, and group them according to the schema. Each group is a possible instance of the given schema, which is mentioned as *semantic block* in the later section. Finally, the representative instances are selected to re-induce the extraction rule for the new page. During the whole process, user-defined schemas are fully used. In addition, our experience with real Web pages shows

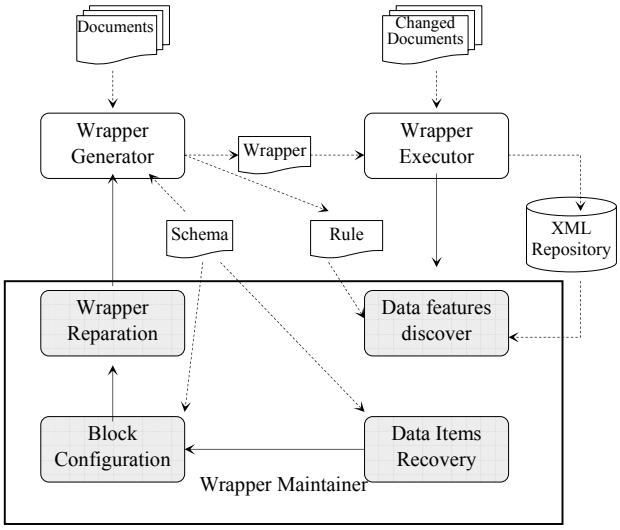



Figure 1: The Architecture of SG-WRAM

that this approach can deal with not only some simple changes, but also most of the complex changes including context shifts, structural shifts [4], and their combinations.

The rest of this paper is organized as follows. Section 2 describes the architecture of the system. Section 3 provides some background of our schema-guided wrapper generator, SG-WRAP. In Section 4 we discuss how to maintain wrappers when page changes. Section 5 reports our intensive experiments on real Web sites. Section 6 discusses related work. In Section 7 we conclude the paper and discuss future research directions.

# 2 System Architecture

Figure 1 depicts the architecture of the implemented system SG-WRAM[16]. The system consists of three major components: *wrapper generator*, *wrapper engine*, and *wrapper maintainer*.

The *wrapper generator* provides a GUI, which allows users to provide an HTML document and an XML schema (DTD), and specify mappings between them. Then the system will generate an extraction rule (wrapper) for this page. The rule extracts data from the page and constructs an XML document conforming to the specified XML schema [14, 15].

The w*rapper engine* provides the execution environments of the generated rules. Given an extraction rule and an HTML page, the engine runs the rule to extract data from the page. In the case where the rule fails, the engine informs the wrapper maintainer to fix the rule automatically.

The *wrapper maintainer* automatically repairs a wrapper that fails to extract correct data after the pages have changed. In this paper we focus on the wrapper-maintenance problem, and existing techniques on wrapper verification [10] to test whether a wrapper stops working.

## 2.1 Schema-Guided Wrapper Generation

The visual supervised wrapper generating approach serves as the base for our techniques of automatic wrapper maintenance. The main idea of the approach is the following. A user defines the structure of her target information by providing a XML schema in the form of a DTD (Figure 2). Given an HTML page, by using a GUI toolkit, the user creates mappings from useful values in the HTML page to the corresponding schema elements. Internally the system parses the HTML page into a DOM[18] tree, and computes the corresponding internal mappings from the HTML tree to the schema tree. Using these mappings the system can generate a tree pattern and output an extraction rule in XQuery expression

An `annotation` of a value used in our approach is a piece of descriptive information that can describe the content of this value. The annotation may lies in another text node near this value in the HTML tree, or can also in the same text node of this value. Table 1 shows the annotations of a few data values (see Figure 3(a)). Note that the annotation of a value could be empty.

Table 1: Annotations for HTML data values

| Data values in HTML page | Annotations |
|---|---|
| May Morning | - |
| Ugo Liberatore | directed by |
| Jane Birkin; John Steiner; Rosella Falk | Featuring |
| 15.38-23.26 | DVD |
| 14.98-18.99 | VHS |

## 2.2 Extraction Rule

The rule in SG-WRAM is an FLWR expression of XQuery [20]. By applying this expression on the HTML page, we can generate an XML document conforming the DTD. In general, in an extraction rule:

- A schema element marked with symbol "+" or "*" (e.g., `VideoList`) corresponds to a clause of "`FOR … RETURN …`".
- Any other element (e.g., `Name`, `Director`, etc.) corresponds to a clause of "`LET … RETURN …`".

The structure of the rule is based on the DTD schema. For each `LET` or `FOR` clause, the system fills in the appropriate XPath[19] on the HTML tree based on the internal mappings. Here's an instance of internal mapping for Figure 3(a), which is transparent to user. D means the data value, HP is the path to this value in HTML tree and SP shows the path to its corresponding DTD element in DTD. Note that the XPath function "contains()" in HP records the annotation for this data value. The first parameter is the path to the annotation starting from the data value, and the second is the value of the annotation.

```
<!ELEMENT VideoList (Video+)>
<!ELEMENT Video (Name, Director, Actors,
Price)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Director (#PCDATA)>
<!ELEMENT Actors (#PCDATA)>
<!ELEMENT Price (VHSPrice, DVDPrice)>
<!ELEMENT VHSPrice (#PCDATA)>
<!ELEMENT DVDPrice (#PCDATA)>
```

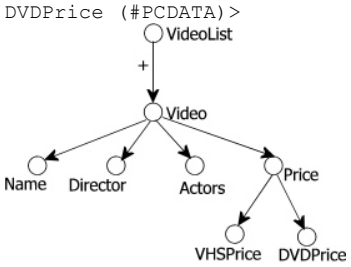



Figure 2: Target DTD schema

*MAPPING(D*: "Ugo Liberatore",
*HP*: ……/text()[0][contains(null,"directed by")],
*SP*: VideoList/Video/Director).

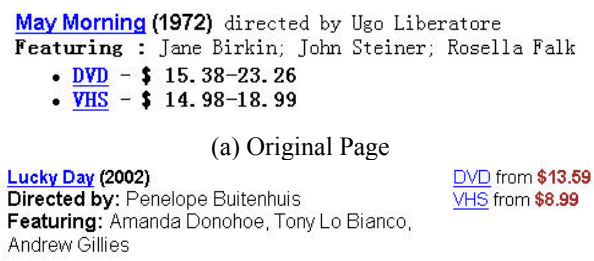
The rule induction algorithm in SG-WRAP starts from the root of the DTD tree and finally computes the extraction rule by recursively calling itself on the children of the current DTD element. The algorithm first finds all the mappings whose *SPs* contain this element. Then it computes a common path of the subtrees in the HTML tree that can include these mappings. A common path is the exactly same parts of the XPath expression. Next, if the element is marked by symbol "*" or "+" in the schema tree, there may be multiple subtrees that contain the input mappings of the current instance, thus the algorithm searches similar subtrees by tentatively generalizing the paths. At last, the generalized common path is used to generate the rule for this element. If the element is not a leaf, the algorithm calls itself recursively for each of its child elements. The rules returned are added to the current rule as subrules. After all these steps, the extraction rule is created.

## 3 Maintaining Extraction Rules

Web pages may change from time to time, and the extraction rules could stop working due to these changes, because even some slight changes in the Web page layout can break a wrapper and prevent it from extracting data correctly. Our later experiment also shows that the format changes often makes the wrappers can hardly extract the correct data items. Thus we want to repair the extraction rules automatically so that they can work for new pages.

The problem of wrapper maintenance includes two subproblems, the first is wrapper verification, and the other is wrapper reparation. Much work has been done on wrapper verification and we focus on wrapper reparation in this paper.

Syntactic features are wildly used for data item recovery in most of the related work, which commonly includes the information of data pattern, string length and so on. But our




(a) Original Page

(b) Changed Page

Figure 3: Sample Web Pages

experience showed that some other data features also plays an important role when maintaining wrappers. E.g. suppose a page contains the following data items: "Our Price: $1.00" and "List Price: $1.20", it's often difficult to distinguish these two, even the information of context are also available, since they have the same syntactic features.

So besides *syntactic features* of data items, we also consider *annotations*, *possible hyperlinks* on data items, and the underlying *schemas*. The syntactic features are certain syntactic conditions of data items, e.g., a street address often has a number and the name of the street. The data pattern gives us an exact description of the data items. We use regular expressions to represent data patterns. As shown in Section 3.1, an annotation of a data item is a descriptive string for this value in the HTML page. For the possible hyperlink on a data item, our experiments show that the information about whether a data item has an associated hyperlink is often preserved after the page changes.

Our approach of wrapper maintenance has four steps.

- *Data-feature discovery*: Data features are computed from the given DTD, the previous extraction rule, and the previous extracted results.
- *Data-item recovery*: Data features are used to recognize the relevant data items in the new page.
- *Block configuration*: We group the recognized data items according to the user-defined schema and the HTML tree structure. Each semantic *block* is an instance of the given schema. (See Section 4.3 for more details.)
- *Wrapper reparation*: The representative instances are selected from the results of block configuration to re-induce the new extraction rule for this changed page.

In this section we discuss these four steps in details. Figure 3 shows the original example page and the changed example page from Yahoo. The original extraction rule fails to extract correct data from the new page because Yahoo has changed its underlying template and the paths to all the data items have changed. We take the following steps to repair the rule automatically.

### 3.1 Item Features Discovery

We consider three important features of each DTD element, represented as a triple ($L$, $A$, $P$):

Table 2: Data Features

| ID | DTD Element | L | A | P |
|---|---|---|---|---|
| 1 | Name | T | NULL | [A-Z][a-z]{0,} |
| 2 | Director | F | Directed by | [A-Z][a-z]{0,} |
| 3 | Actors | F | Featuring | [A-Z][a-z]{0,}(.)* |
| 4 | VHSPrice | F | VHS | [$][0-9]{0,}[0-9](.)[0-9]{2} |
| 4 | DVDPrice | F | DVD | [$][0-9]{0,}[0-9](.)[0-9]{2} |

- *L*: A Boolean value (TRUE or FALSE) to indicate if the data item *d* corresponding to this element has an associated hyperlink.
- *A*: An annotation of the data value *d*, as described in Section 3.
- *P*: A data pattern, which is a regular expression for this data value.

For instance, Table 2 shows the tuples for the DTD elements "Name", "Director" and "Actor". It has a row for each element with a unique ID.

The system computes the entries for each DTD element as follows. The value of L is straightforward: its value is TRUE if there is a hyperlink associated to the corresponding data element in the HTML page, and FALSE otherwise. The annotation is recorded in the extraction rule with the XPath function of "*contains()*"(see section 3). And for the data pattern of each DTD element, we could generate it by studying several example pages with the same structure [6], using machine learning techniques, or applying pattern-extraction techniques [3].

### 3.2 Data-Item Recovery

In this step, we traverse the new HTML tree following the depth-first traversal (DFS) order. For each leaf node *n*, if the system finds that a data value may be an annotation of an item, it tries to find the corresponding value of this item. Otherwise, we check the item table to see if it satisfies the three conditions (features) of an item row *r*. That is, if r.L = "TRUE" ("FALSE"), then node *n* does (not) have an associated hyperlink. The annotation of node *n* should be the same as *r.A(*it can be null*)*. The string of this value is recognized by the regular expression (data pattern) of *r*. In this case, the leaf node n is called an *instance* of the item *r*. Node *n* is expected to be an instance of the corresponding DTD element. For example, if a node string *n* is an instance of the Item 2 in Table 2, then *n* should have an annotation "Directed by", it has no hyperlink, and is accepted by the data pattern, "*[A-Z][a-z]{0,}(.)**". In this case, this node is one of the target nodes of Item 2. We create an item-instance list in this step, which is an instance array following the depth-first traversal order, and it is maintained in the process of the data-item recovery.

During this step, if the annotation of an item changed in the new pages, it will be treated as a different item. E.g. it's a great task for a program to decide whether "Our Price: $1.00" and "Price: $1.00" are corresponding to the same DTD element. What's more, because our algorithm bases on a given schema, we don't discuss newly add data items in this paper for the reason that they do not appear in the given schema.

Table 3: Item Instance Table

| No. | ID | PATH |
|---|---|---|
| 1 | 1 | …**table[1]**/tr[0] /td[1]/span[0]/b[0]/a[0]/text()[0] |
| 2 | 2 | …**table[1]**/tr[0]/ /td[1]/span[1]/text[contains( /preceding-sibling::b[0],"Directed by")] |
| 3 | 3 | …**table[1]**/tr[0]/ /td[1]/span[2]/text()[contains( /preceding-sibling::b[0],"Featuring")] |
| 4 | 1 | …**table[2]**/tr[0] /td[1]/span[0]/b[0]/a[0]/text()[0] |
| 5 | 2 | …**table[2]**/tr[0]/ /td[1]/span[1]/text[contains( /preceding-sibling::b[0],"Directed by")] |
| 6 | 3 | …**table[2]**/tr[0]/ /td[1]/span[2]/text()[contains( /preceding-sibling::b[0],"Featuring")] |

Meanwhile, some noises who repeatedly occur in most of the semantic blocks or all of the semantic blocks, e.g. pages about E-book from www.amazon.com contains a sentence of "Click here for more info" in most of the blocks, it's often recognized as a possible data item in the course of item recovery, are removed from the item instance table. In fact, these noises are some parts of the underlying template of the pages.

At last, we get the item instance list as shown in Table 3. The ID here is the ID of an item in the Item Table 2. For the limited space, we only provide the results on the first 3 items.

### 3.3 Block Configuration

After recovering all the possible data items, we want to find out the underlying organization of these data items. We find that the data items are grouped in different semantic blocks. We first introduce a few important notations. An HTML document can be viewed as containing a set of semantic blocks, and each semantic block is a fragment of the HTML tree that conforms to the user-defined schema. A semantic block includes a set of instances of schema elements. It is a subtree, or several sibling subtrees that include all the instances of schema elements. Each instance of schema element is a row of the Item Instance Table (Table 3). So in this step we construct *semantic blocks* from the new HTML page.

A semantic block is called an *atomic semantic block* if it satisfies the following conditions:

- It is a subtree or set of sibling subtrees; and
- The occurrences of data values conform to the definition of schema.

Intuitively, an atomic semantic block is the minimum extractable unit with which we can extract an XML document conforming to the schema, it's similar to the instances user selected in wrapper generation. Comparing

with the schema, a *match* between a block *A* and the schema can be one of the following three cases:

- *Over match*: There is at least one item *i* in the schema that occurs at least twice in block *A*.
- *Full match*: Block *A* contains all items of the schema and satisfies the constraint of each item in the schema, such as '+' or '*', '?' etc.
- *Partial match*: Block *A* contains a subset of items of the schema.

In the block-configuration step, we first identify the level of block configuration in a top-down manner. At the level *k* of the new HTML tree, each sub-tree is viewed as a possible block. *Subtree weight* of a subtree is the number of possible data items in this subtree. As most of the Web pages containing interesting data come from underlying template, our observation shows this guarantees that amost all the data are always located in a big subtree or several subtrees, although there's often complex structure in these subtrees. We use this Subtree Weight for excluding some low-weighted noise subtrees.

After classifying all possible blocks at a level, the number of blocks in each kind of matches is counted without considering the non-important subtrees. The group *R* with the largest number of a special match is used to decide the next step.

- If *R* is the full-match group, return all the blocks that are full matched.
- If *R* is the partial-match group, merge sub-trees in the level *k*-1.
- If *R* is the over-match group, turn to the level *k*+1 and continue to block configuration.

So if the blocks in the changed pages can fully match with the schema, this step will stop at the level where the system find all the full matched blocks.

But in many cases, we cannot find the right level where we can get all the full matched blocks, e.g. the changed page contains only parts of the data items from the original page. So at a certain step the system finds that data items are scattered in several partial matched blocks, while they should be in the same block. So we should merge them. First, we merge two sibling subtrees. If the result is still a partial match, we continue merging sibling subtrees with the same parent. We repeat this step, until the algorithm stops at level *n*, where we find that the algorithm get too many over-matched blocks at level *n-1* and get too many partial-matched blocks at level *n+1*.

Thus in the *Item Instance Table* (Table 3), the items from row 1 to row 3 are finally decided in the same block and those from row 4 to row 6 are in another block.

### 3.4 Wrapper Reparation

The results of block-configuration are a set of semantic blocks. Next, we pick up a representative block as an instance to construct mappings from the data values in the new HTML page to the DTD schema. Then we can re-induce the new extraction rule by calling the wrapper generator.

In our running example, we choose the first block in the Table 3 to construct the internal mappings as follows:

*M1'*(*D*: "Lucky Day",
 *HP*:…/**table**/tr[0]/td[1]/span[0]/b[0]/a[0]/text()[0],
 *SP*: VideoList/Video/Name )
*M2'*(*D*: "Penelope Buitenhuis",
 *HP*: …/**table**/tr[0] /td[1]/span/text()[contains(/preceding-sibling::b[0],"Directed by")],
 *SP*: VideoList/Video/Director ),
*M3'*(*D*: "Amanda Donohoe, Tony Lo Bianco, Andrew Gillies",
 *HP*:…/**table**/tr[0]/td[1]/span/text()[contains(/preceding-sibling::b[0],"Featuring")],
 *SP*: VideoList/Video/ Actors),
……

By running the rule induction algorithm in SG-WRAP, we generate the new extraction rule.

Of course, we also face the risk of a possible bad instance making some of the data items cannot be successfully extracted, and a few of data items are missed when the repaired wrapper applies to other pages with the similar structure. E.g. dealing with a set of pages from the same search engine, a few data items in some pages are perhaps missed because of some tiny difference in structures comparing with the common structure. If it's needed, the SG-WRAP system will automatically select other representative instances from the results of block configuration to refine the repaired wrapper. Then the rules are automatically integrated. Since our extraction rule uses XQuery expression, it's easy to integrate two extraction rules. The process is the following:

- If the rule from new instances contains new predicates for a certain data item, the predicates are added into the extraction rule.
- If the new rule contains a new path to a data item, the path is added into the extraction rule.

## 4 Experiments

To evaluate our approaches to wrapper maintenance, we conducted a number of experiments on real Web pages. We monitored 16 Web sites October 2002 to May 2003. The sites are listed in Appendix A. For each Web site, we periodically archived some pages of the same URL or pages from the same query.

In order to increase the chance of getting changed pages from the real Web sites, we collected several sets of pages with different structures from each site, and of course, we need different wrappers for these different structures.

For each set of collected pages we do the following:

(1). Run the SG-WRAP system on the earliest pages and generate a wrapper for each set of pages.
(2). Apply the initial wrapper to the newly collected Web pages. By manually checking how many data items corresponding to the elements in the DTD can

Table 4: Initial wrapper on changed pages

| Name | R% | P% | Item Number | Page Number |
|---|---|---|---|---|
| 1Bookstreet | 82.54 | 100 | 6 | 12 |
| Allbooks4less Book | 0 | - | 4 | 15 |
| Amazon Book (search) | 40.49 | 100 | 6 | 15 |
| Amazon Magazine | 20.01 | 100 | 5 | 15 |
| Barnesandnoble Book | 0 | 100 | 5 | 15 |
| CIA Factbook | 0 | 100 | 10 | 5 |
| CNN Currency | 50.00 | 100 | 6 | 15 |
| Excite Currency | 42.86 | 100 | 11 | 18 |
| Hotels Hotel | 0 | - | 4 | 15 |
| Yahoo Shopping Video | 0 | - | 6 | 15 |
| Yahoo Quotes | 0 | - | 6 | 10 |
| Yahoo People Email | 0 | - | 3 | 10 |

(3). be correctly extracted, we make certain if a page has changed.

(4). For each set of changed pages, through our approach of wrapper maintenance, we get a repaired wrapper. Then the repaired wrapper is applied on the changed pages.

### 4.1 Evaluation Metrics

We first define the following assistant parameters:

- *CN*: number of correct data items that should be extracted in a page;
- *EN*: number of extracted data items by the wrappers;
- *CEN*: number of the correctly extracted data items by the wrappers.

We use two metrics, *Precision* and *Recall,* to evaluate the results of our algorithm of wrapper maintenance.

- *Recall (R)*: proportion of the correctly extracted data items of all the data items that should be extracted. It can be presented as "*R = CEN/CN*".
- *Precision (P)*: proportion of the correctly extracted data items of all the data items that have been extracted. It can be presented as "*P = CEN/EN*".

### 4.2 Analysis of Changed Pages

After applying the initial wrappers on the newly collected pages, we find those who make poor performances on *Recall* and *Precision* and manually check if they have taken format changes. We find some of the pages changed from the following sites, shown in Table 4. The first column of the table lists the wrapper name. The other columns list the value of *R, P*. The last column
shows the number of changed pages of the sources we selected for the experiment, which is used in the latter tests.

The symbol of "-" means that the value of P isn't computable. It's because the initial wrappers cannot get any data item from the changed pages, thus the value of EN and CEN are all 0, which makes the value of Precision P = CEN/EN not computable.

Table 5: Wrapper maintenance

| Name | R%(IR) | P%(IR) | R%(EX) | P%(EX) |
|---|---|---|---|---|
| 1Bookstreet | 98.67 | 71.26 | 100 | 100 |
| Allbooks4less Book | 75 | 32.69 | 75 | 51.34 |
| Amazon Book (search) | 83.05 | 36.3 | 83.05 | 90.74 |
| Amazon Magazine | 100 | 60.15 | 100 | 100 |
| Barnesandnoble | 78.72 | 43.13 | 78.72 | 100 |
| CIA Factbook | 100 | 100 | 100 | 100 |
| CNN Currency | 100 | 100 | 100 | 100 |
| Excite Currency | 100 | 100 | 100 | 100 |
| Hotels Hotel | 50 | 35.61 | 50 | 41.87 |
| Yahoo Shopping | 100 | 51.49 | 100 | 92.86 |
| Yahoo Quotes | 100 | 100 | 100 | 100 |
| Yahoo People | 100 | 53.54 | 100 | 100 |

Here're some details: On `1Bookstreet`; `Amazon Book`, `Amazon Magazine` and `Excite Currency`, the initial wrapper can still extract some data items from the changed pages.

`CIA Factbook` changes the structure and still remaining the non-table structure. And on the other example sites, the templates of the sites are changed, making the initial wrapper invalid completely. E.g. `Yahoo Quotes` changes the structure from a complex table to non-table structure.

### 4.3 Effectiveness Test

After identifying those changed pages, the system automatically computes data features and takes the steps of wrapper maintenance. In this test, we re-induce extraction rules from a single changed page, the wrapper is only refined with the instances within the page, without being refined for the page set. Then we run the maintenance algorithm automatically and check the results by human.

#### 4.3.1 Basic Results

Table 5 is the results of wrapper maintenance. For each wrapper, we compute the following metrics: Recall, Precision after the step of *item recovery*, and the corresponding values after the system repaired the extraction rule and apply the new wrapper on the changed pages.

Among the results, `CIA Factbook`, `CNN Currency`, `Excite Currency`, `Yahoo Quote` get perfect results on the 4 metrics. `CIA Factbook` benefits from the fact that all the data features of the date items are perfectly preserved, although the Web site has changed its underlying template completely. The data features here means all the 3 features of data pattern, annotation and

hyperlink flag. As to CNN Currency, Excite Currency, Yahoo Quote, all of them have all the data items in a simple table, and all the data items of them are purely digit. What's more, the annotations of them are perfectly preserved, that is, the table head are the same. So the system can precisely find out all the data items without any noises in all the steps.

On 1Bookstreet, all the data features are preserved, so the item "You Save" with small structure change on it can be located from the changed pages. Because the item of "Availability" has multi expressions in the sites, and the example pages cannot contain all the patterns of these expressions, several corresponding item instances are missed when taking the item recovery step. But during the step of rule reparation, the common path computed in the extraction rule helps find these item instances, so the value of R%(EX) get 100% in the extracted results.

On Amazon Book, annotation of an item of "Our Price" changed to "Buy New", and the system treated them as different data items, so repaired extraction rule did not contain this data item. Meanwhile, because the pages contain another big subtree showing "the most popular books", which made the system get 3 blocks from them, while these blocks are not wanted. So P%(EX) fails to get a satisfactory value. So does Barnesandnoble, the system failed to recognize the data item of "Availability" because of changed data pattern.

On Hotels and Allbooks4less, both of them get poor performance on almost all the metrics. Although most of the data patterns of them are not changed, but almost all the other features, annotation and hyperlink flag, fail to be preserved, so during the step of item recovery, too many noises are falsely recognized and can not been effectively excluded in the latter steps. Thus the system can hardly find correct blocks from the results of block configuration, and the repaired extraction rule cannot work well correspondingly.

What's more, we find that the value of P%(IR) are much higher than the P%(EX) on almost all the examples, which means the step of block configuration has excluded many of the noise subtrees.

We have discussed that these examples have cover a large scale of representative structure changes, we conclude that our maintenance algorithm has high practicability. Meanwhile, our experiment also shows the usage of annotation and hyperlink flag brings much advantage for wrapper maintaining.

#### 4.3.2 Extensive Discussion on Results

In the previous section we know that annotation and hyperlink takes an important role in wrapper maintenance with the help of traditional usage of data pattern (syntactic features). Figure 4 indicates how the data features of annotation and hyperlink work in our approach.

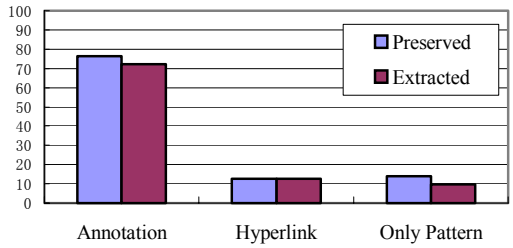



Figure 4: Discussion on Data Features

Firstly, we find about 80% of the data items have annotations, and almost all of this data items can be successfully extracted by repaired wrapper with cooperation with data pattern.

Although fewer data items have the data feature of hyperlink, they are mostly accorded with the item "Name" of a book or other merchandise, those data items can all be extracted. Otherwise, we'll have to rely on the syntactic features of them. The "*Only Pattern*" part in Figure 4 illustrates the case. The biggest trouble taken by this case is too many noises may be created during the step of item recovery, making the wrapper cannot be successfully maintained. In fact, as to the examples of Hotels in Table 5, the system has to recognize 3 of 4 data items by pure data pattern, which produced too much noises and greatly affect the following steps.

## 5 Related works

There are two aspects on wrapper maintenance: *change detection* and *wrapper reparation*. Kushmerick [10] focused on wrapper verification, i.e., change detection, and presented an algorithm RAPTURE for solving this problem. RAPTURE compares the pre-verified label with the label output by the wrapper being verified. Specifically, it compares the value of various numeric features of the strings comprising the label output by the wrapper. Then, an overall probability that the wrapper is correct for the page is computed. If the overall probability is less than a user-defined threshold, the page is considered to be unchanged; otherwise, it is considered to have changed. But if the generic features of some data fields are similar to the right one, their system cannot detect the change.

[12] addressed the both aspects of wrapper maintenance: changed detection and wrapper reparation. For the first problem, they applied machine learning techniques to learn a set of patterns that can describe the information that is being extracted from each of the relevant fields. They used the starting and ending strings as the description of the data field. If the pattern describes statistically the same proportion of the test examples as the training examples, the wrapper is considered to be working correctly. Next, the wrapper re-induction algorithm takes a set of training

examples and a set of pages from the same source, and uses machine learning techniques to identify examples of the data field on the new pages. This method produces too many candidates of data fields. Many of them are noises. The process of clustering the candidates for each data field does not consider the relationship of all data fields. Furthermore, the top ranked cluster may not include all correct candidates. If we only use the examples in top ranked cluster for the re-induction, the new rule may be unfitted for all pages.

Compared with the above methods, our approach has several advantages:

(1). It can detect more changes than others. For instance, suppose we want to extract the *Title*, *List Price*, and *Our Price* from the following table.

| Title | List Price | Our Price |
|---|---|---|
| Data on Web | $29.00 | $23.00 |
| Java Programming | $59.00 | $49.00 |

If the page is changed and the column of *Our price* is put before the column of *List Price*:

| Title | Our Price | List Price |
|---|---|---|
| Data on Web | $23.00 | $29.00 |
| Java Programming | $49.00 | $59.00 |

The methods proposed in [10, 12] cannot detect this kind of changes, because the generic features and data patterns of *List Price* and *Our Price* are the same. Notice our approach considers the annotations of data items. Thus by applying the extraction rule to the changed page, only the titles of two books are in the extraction result. After checking the result, the system finds that nothing is extracted for *List Price* and *Our Price,* and thus knows that the wrapper failed working

(2). In the phase of item recovery, by using our data features (data patterns, annotations, and hyperlink flags), our approach produces less unrelated candidates than that of [12].

(3). By using the schema to group the data items, our approach is effective to reduce the noise data items and get right new mappings to re-generate the extraction rule.

## 6 Conclusion

In this paper, based on our previous work of schema-guided wrapper generator SG-WRAP, we propose a novel schema-guided approach to the issue of wrapper maintenance. By using the preserved data features in the changed pages, our approach can identify the locations of the desired data items in the changed pages, and automatically generate new wrappers correspondingly. Our intensive experiments with real Web pages showed that the proposed approach can effectively maintain wrappers to extract desired data with high accuracies.

## 7 References


[1] Ashish N, Knoblock C A. Wrapper generation for semi-structured Internet sources. SIGMOD Record, 1997, 26(4): 8-15.

[2] Baumgartner R, Flesca S, Gottlob G..Visual Web Information Extraction with Lixto. In Proceedings of the Very Large Data Bases; 2001, 119-128.

[3] Brin S. Extracting patterns and relations from the world wide web. In International WebDB Workshop, Valencia, Spain, pages 172-183, 1998.

[4] Chidlovskii B. Automatic repairing of Web Wrappers. In 3rd International Workshop on Web Information and Data Management, 2001, 24-30.

[5] Doorenbos R, Etsionoi O, Weld D S. A scalable comparison-shopping agent for the world-wide-web. In Proceedings of the First International Conference on Autonomous Agents, 1997, 39-48.

[6] Gupta A., Harinarayan V., Quass D., and Rajaraman A. Method and apparatus for structuring the querying and interpretation of semistructured information. United States Patent number 5,826,258, 1998.

[7] Hammer J, Brenning M, Garcia-Molina H, Nestorov S, VassalosV, Yerneni R. Template-based wrappers in the TSIMMIS system. In Proceedings of ACM SIGMOD Conference, 1997, 532-535.

[8] Knoblock C A, Lerman K, Minton S, Muslea I. Accurately and Reliably Extracting Data from the Web: A Machine Learning Approach. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2000, 23(4): 33-41.

[9] Kushmerick N, Weil D, Doorenbos R. Wrapper induction for information extraction. In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1997, 729-735.

[10] Kushmerick N. Regression testing for wrapper maintenance. In Proceedings of AAAI, 1999,74-79

[11] Kushmerick N. Wrapper verification. World Wide Web Journal, 2000, 3(2): 79-94.

[12] Lerman K. and Minton S.. Learning the common structure of data. In AAAI2000.

[13] Liu L, Pu C, Han W. XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources. In Proceedings of ICDE, 2000, 611-621.

[14] Meng X F, Lu H J, Wang H Y, Gu M Z. SG-WRAP: A Schema-Guided Wrapper Generator. Demonstration in ICDE, 2002, 331-332.

[15] Meng X F, Lu H J, Wang H Y, Gu M Z. Schema-Guided Data Extraction from the Web. Journal of Computer Science and Technology (JCST), 2002,17(4).

[16] Meng X F, Wang H Y, Hu D D, SG-WRAM: Schema Guided Wrapper Maintenance, Demonstration in Proceedings of ICDE, 2003, 750-752.

[17] Sahuguet A, Azavant F. Building Light-Weight Wrappers for Legacy Web Data-Sources Using W4F. In Proceedings of VLDB, 1999, 738-741.

[18] DOM Document Object Model (DOM) Level 2 Core Specification http://www.w3.org/TR/DOM-Level-2-Core

[19] XML Path Language (XPath) 2.0, http://www.w3.org/TR/xpath20/

[20] XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/