ISSN:0975-9646

# DEUDS: Data Extraction Using DOM Tree and Selectors

Vinayak B. Kadam , Ganesh K. Pakle

*Department of Information Technology,*
*SGGS IE & T, Nanded,*
*Maharashtra, India-431606*

***Abstract*— Web data analysis applications such as extracting mutual funds information from a website, daily extracting opening and closing price of stock from a web page involves web data extraction. Every time you need analyze data, you need to visit number of web sites. It is very time consuming process to construct wrapper to visit those sites and collect data. In this paper, we propose technique called DEUDS, a page level data extraction system that automatically discovers extraction pattern from web pages for selected data section and extracts data. DEUDS uses visual cues to identify data records while ignoring noise items such as advertises and navigation bars.**

***Keywords*— DOM Tree, CSS selector, semi structured web pages and Web data extraction.**

## I. INTRODUCTION

Different Web sites contain information on various topics in various formats. Large amounts of effort are often required for a user to manually locate and extract data of interest from the Web pages. For example, great efforts are needed to generate Web information gathering crawlers, comparison-shopping agents, and news bots, etc. A previous approach for extracting structured data from web pages was to write programs, called “wrappers” or “extractors” or “crawler”, to extract the contents of the Web pages based on a priori knowledge of their format e.g. tag tree structure. In other words, we have to observe the structure of web page and write programs for each Web site. It is important to note that web page belonging to same web site contains same structure or template. Templates is the model for all pages, occur quite fixed as opposed to data values which changes across pages.

However, programming wrappers require manual coding which generally requires labour. And also, the format of Web pages is often subject to change, maintaining the wrapper can be expensive and impractical. Extracting structured data enables us to integrate data from multiple web pages to pose more complex queries and to provide value-added services, e.g., comparative shopping. Lots of the existing work in extracting data is based on identifying repeated patterns. We differentiate the existing approaches based on where they look for these patterns and how they use them in data extraction. Wrappers are generated automatically by number of researchers, e.g. WIEN [3], Softmealy [2], Stalker [4] etc.

Existing approaches are also depends on template of web page. Finding structure of template requires multiple web pages or a single web page containing multiple pages as input. When multiple sample web pages are given, the extraction target aims at page-wide information (e.g. RoadRunner [5] and EXALG [8]). RoadRunner assumes that site generation is process of encoding the original database content into strings of HTML code. As a result, data extraction is thought as a decoding process.

EXALG deduces the template and uses it to extract the set of values from the encoded pages as an output. EXALG detects the unknown template by using the two techniques differentiating roles and equivalence classes. When single web pages are given, the extraction target is usually targeted to record wide information (e.g., IEPAD [9], DELA [7], and DEPTA [6]), which involves issue of record-boundary detection. IEPAD generates extraction patterns from unlabeled Web pages [31]. This method assumes that if a Web page contains multiple data records to be extracted, they are displayed regularly using the same template. if the page is well encoded repetitive patterns can be found. In IEPAD PAT trees data structure is used, which is a binary suffix tree to discover repetitive patterns in a Web page. FiVaTech [20] proposes method to extract template data from web page which contains fixed template.

DELA uses two consecutive steps to generate wrappers. First, Data-rich Sections are identified from Web pages by comparing the DOM trees for two Web pages. Second, repeated patterns are found using suffix trees.

DEPTA (Data Extraction based on Partial Tree Alignment): DEPTA founds repeated substring by comparing only adjacent substrings with starting tags having the same parent in the HTML tag tree.

Approaches [10, 6, 11, 12, 13, and 14] use the tag tree representation of a web page to identify repeated patterns. Before performing extraction these tools turn web page into tag tree. This hierarchal representation of the source code is very useful. As we know that, the tag tree was designed for the browser to use when displaying the page, and unfortunately there may be some mismatched tag.

W4F (Wysiwyg Web Wrapper Factory) is a Java toolkit to generate Web wrappers [13]. Wrapper development process in W4F consists of three independent steps: retrieval, extraction and mapping step. In the retrieval phase, document to-be processed is retrieved and fed to an HTML parser that constructs a parse tree. In the extraction phase, extraction rules are applied on the parse tree to extract information. Mapping phase is used to export NSL structures. In XWRAP [14] wrapper generation process includes two phases: structure analysis, and source-specific XML generation. First, XWRAP reads, and generates a tree-like structure of the web page. Then system identifies data regions. In the second phase, the system generates a XML template file, and then constructs a source-specific XML generator, XWRAP.

Approaches [17, 16, 15] identify visually repeated patterns using additional information displayed on a rendered page. However, they all have limitation: they depend on direct access to either the source code or the tag tree. The approach in [18] is the first that uses visual features for data record extraction. But, this approach has several limitations like how it deals with noise items on a web page. Approaches [18, 16, 6, 15] divides page into sections, referred to as the data-rich section, which contains all of the data records. But, identifying the data section can be problematic because, unwanted noise items incorrectly included in, the data section. May be it is possible to identify the wrong sub-section of the page entirely.

In this paper, novel approach is presented to extract data, which uses visual cues and CSS Selector for attributes of DOM tree to construct pattern. This paper proposes a three-step strategy to solve the problem of data extraction.

1) In this step, page is divided into sections. Section is nothing but the part of page containing useful data. Some technique uses MDR [10] for dividing page into section. We are using visual cues to find data records. Visual information helps in two ways:

a) It enables to identify gaps that separate data records, which helps to segment data records correctly because the gap within a data record (if any) is typically smaller than that in between data section.
b) System identifies data records by analyzing HTML tag trees or DOM trees [19]. A tag tree is built by following the nested tag structure in the HTML code. However, we have to take care of missing or ill formatted tags. The visual or display information can be obtained after the web page is rendered by a Web browser, it also contains information about the hierarchical DOM tag tree structure. In our work, instead of analyzing the HTML code, visual information i.e. the locations on the screen at which tags are rendered is utilized to infer the structural relationship among tags and to construct a tag tree and to get selectors. As long as the browser is able to render a page correctly, its tag tree can be built correctly.

2) In second step, grammar or more precisely pattern is generated using DOM tree and selectors from DOM Tree.
3) In third step, data is extracted from web pages using the grammar generated in second step.

Our three step approach called DEUDS (Data Extraction Using DOM Tree and Selectors) is different from existing approaches in two different ways. First, how it identifies data section and second, how it construct pattern to extract data. We propose a visual approach which identifies sections of data, and also a single data item, which is a basic content block in a data record. And also, our visual approach directly retrieve positional information and visual features of each item on the page, avoiding the need to interpret increasingly complex HTML source code and tag trees. Our proposed technique DEUDS presents number of advantages over existing systems.

a) Here, User can generate extraction rules with few mouse clicks.
b) Identifying data region using visual cues is very simple, because cost of comparing DOM tree is reduced.
c) It provides separation of extraction pattern generation and wrapper generation. This separation allows wrapper to use new extraction rules.

The rest of the paper is organized as follows: The fundamentals architecture of our proposed DEUDS approach is presented in Section 2. Section 3 presents details of web page renderer. In Section 4 details of section selector are provided. Section 5 gives details of how to construct pattern and how to retrieve data with that pattern. Experimental evaluation is reported in Section 6. Related work is presented in section 7 and finally, Section 8 concludes the paper.

## II. System Overview

The system DEUDS includes three components, a web page renderer which accepts an input Web page. After a web page is displayed using browser, DOM tree creator create DOM tree. Section selector divides web page into the Data sections, from which you can select particular record or whole data section. Pattern generator generates patterns based on data section selected. Here patterns generated are relative not absolute, so no need to worry about the change of structure of web page.

The pattern generator includes a pattern generation from attributes, and a pattern validator. The pattern generator retrieves patterns discovered in a Web page. The graphical user interface is used by users to view the data extracted by extraction rule. As user selects extraction rule conforming to his information desire, the extractor phase can use it to extract information from similar web pages.

**Web Page Renderer** is the first component, it performs three tasks. (i) It accepts URL by user issues an http request and fetches corresponding document. This web page is used to derive grammar. (ii) It cleans bad and ill formatted html tags. (iii) It generates DOM tree from retrieved web page.

**Section Selector** divides input web page in to data section. Here we dividing page into different sections like list section, single valued section, multi valued section etc. it performs three tasks. (i) Identifying data section in retrieved web page. (ii) Identifying important semantic tokens and attributes and there logical path in DOM tree. (iii) Identifying useful hierarchical structure.

**Pattern Generator** is responsible for generating extraction pattern to extract data of interest. It performs three tasks. (i) It generates pattern from token and attributes retrieved in section selector. (ii) Pattern is validated based on uniqueness of pattern in document. (iii) Data is extracted using pattern.

Fig 1 shows the overview of the system. Web page from which we want to extract data is given as input to DEUDS. Web page renderer visualize input page from which we can see the various features of page or more precisely different region containing data to be retrieved. Section selector selects data region called section using visual cues like boundary, width etc. when we select section, pattern generator create pattern by using selectors. Data extractor uses pattern derived by pattern generator to extract data and store this data in the database. In next few sections, we will see the details of each sub system.

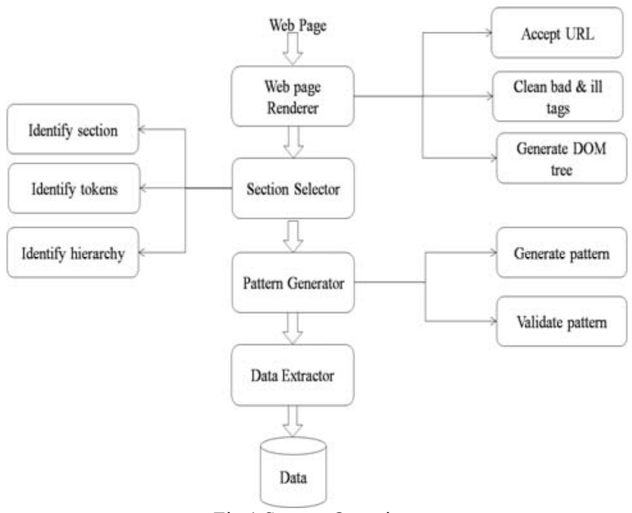


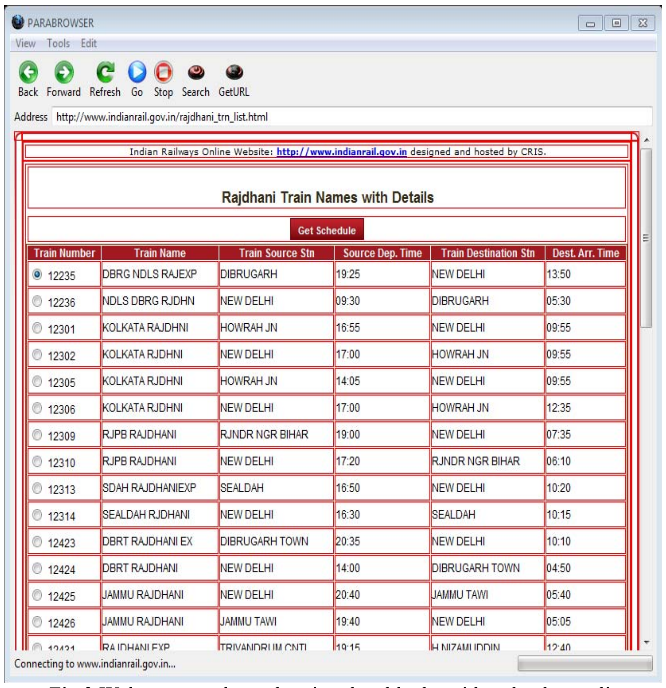
Fig 1 System Overview

## III. Web Page Renderer

The system DEUDS includes three components, a web page renderer which accepts an input Web page. After a web page is displayed using browser, DOM tree creator create DOM tree. Section selector divides web page into the Data sections, from which you can select particular record or whole data section. Pattern generator generates patterns based on data section selected. Here patterns generated are relative not absolute, so no need to worry about the change of structure of web page.

### *A. URL Acceptor*

Your Here we are giving URL of page. When browser receives URL, browser sends http request to access requested document from web. We know that web page is made of the tag tree and the Cascading Style Sheet (CSS) of the page. A layout engine generates page from nodes of tag tree, according to the styles contained in the CSS. This process, called rendering, draws a rectangular box around the minimum boundary of each visible node on the page. We refer to each box as a visual block. The position of each visual block is represented by its four borders in the four directions on the two-dimensional plane. The outer block contains many inner blocks. The inner block which does not contains any further inner blocks is called basic block, which may contains data value. Basic blocks are shown by red color outline. From fig 2 we can see that outer blocks contain many inner blocks.



Address http://www.indianrail.gov.in/rajdhani_trn_list.html

Indian Railways Online Website: http://www.indianrail.gov.in designed and hosted by CRIS.

Rajdhani Train Names with Details

Get Schedule

| Train Number | Train Name | Train Source Stn | Source Dep. Time | Train Destination Stn | Dest. Arr. Time |
|---|---|---|---|---|---|
| ● 12235 | DBRG NDLS RAJEXP | DIBRUGARH | 19:25 | NEW DELHI | 13:50 |
| ○ 12236 | NDLS DBRG RJDHN | NEW DELHI | 09:30 | DIBRUGARH | 05:30 |
| ○ 12301 | KOLKATA RAJDHNI | HOWRAH JN | 16:55 | NEW DELHI | 09:55 |
| ○ 12302 | KOLKATA RJDHNI | NEW DELHI | 17:00 | HOWRAH JN | 09:55 |
| ○ 12305 | KOLKATA RJDHNI | HOWRAH JN | 14:05 | NEW DELHI | 09:55 |
| ○ 12306 | KOLKATA RJDHNI | NEW DELHI | 17:00 | HOWRAH JN | 12:35 |
| ○ 12309 | RJPB RAJDHANI | RJNDR NGR BIHAR | 19:00 | NEW DELHI | 07:35 |
| ○ 12310 | RJPB RAJDHANI | NEW DELHI | 17:20 | RJNDR NGR BIHAR | 06:10 |
| ○ 12313 | SDAH RAJDHANIEXP | SEALDAH | 16:50 | NEW DELHI | 10:20 |
| ○ 12314 | SEALDAH RJDHANI | NEW DELHI | 16:30 | SEALDAH | 10:15 |
| ○ 12423 | DBRT RAJDHANI EX | DIBRUGARH TOWN | 20:35 | NEW DELHI | 10:10 |
| ○ 12424 | DBRT RAJDHANI | NEW DELHI | 14:00 | DIBRUGARH TOWN | 04:50 |
| ○ 12425 | JAMMU RAJDHANI | NEW DELHI | 20:40 | JAMMU TAWI | 05:40 |
| ○ 12426 | JAMMU RAJDHANI | JAMMU TAWI | 19:40 | NEW DELHI | 05:05 |
| ○ 12431 | RAJDHANI EXP | TRIVANDRUM CNTL | 19:15 | H NIZAMUDDIN | 12:40 |

Connecting to www.indianrail.gov.in...

Fig 2 Web page renderer showing data blocks with red color outline

### *B. REPAIRING ILL TAGS*

As soon as document is fetched, process of repairing bad or ill formatted tag begins. This process inserts missing tags, removes useless tags e.g. tag starting with !pr is end tag having no start tag. It also checks proper nesting of all tags. This process of cleaning document is applicable to all html pages.

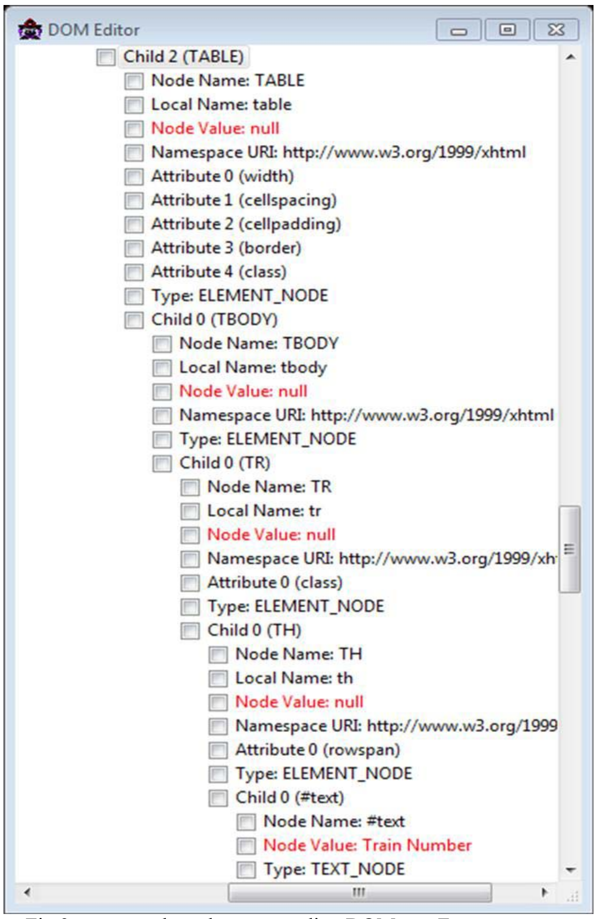
### *C. Building DOM Tree*

After bad and ill formatted tags are removed from web page source code, we can use this code to build DOM tag tree. Each html element consists start tag, optional attributes, optional embedded content, and end tag. DOM API is used to construct the tree for web page. Each page contains zero or one doc type nodes, one root element node, and zero or more comments or processing instructions; the root element serves as the root of the tree for the page. Parser converts source document into syntactic token, from this token tree is generated. Fig 3 shows sample html code and DOM tree for that code. From fig 3 we can see how all elements, corresponding attributes, and content is added to DOM tree.

## IV. Section Selector

This section focuses on the segmenting the Web page to identify individual data section. In this step we do not extract any data records. When we select section, simultaneously attributes of selected sections are also retrieved from DOM tree. Here we constructed DOM tree of rendered page using SWT. We already seen that web page is made up of basic blocks, as result data section is also made up of these blocks. Data section may contain many basic blocks, which actually contains data values. We designed java script to highlight the selected data section or selected basic block.

```
<TABLE width="100%" border="0" cellpadding="0" cellspacing="1" class="table_border">
<TR class="heading_table_top">
<TH ROWSPAN = 1>Train Number</TH>
<TH ROWSPAN = 1>Train Name</TH>
<TH ROWSPAN=1>Train Source Stn</TH>
<TH ROWSPAN=1>Source Dep. Time</TH>
<TH ROWSPAN=1>Train Destination Stn</TH>
<TH ROWSPAN=1>Dest. Arr. Time</TH>
</TR>
```



```
DOM Editor
Child 2 (TABLE)
    Node Name: TABLE
    Local Name: table
    Node Value: null
    Namespace URI: http://www.w3.org/1999/xhtml
    Attribute 0 (width)
    Attribute 1 (cellspacing)
    Attribute 2 (cellpadding)
    Attribute 3 (border)
    Attribute 4 (class)
    Type: ELEMENT_NODE
    Child 0 (TBODY)
        Node Name: TBODY
        Local Name: tbody
        Node Value: null
        Namespace URI: http://www.w3.org/1999/xhtml
        Type: ELEMENT_NODE
        Child 0 (TR)
            Node Name: TR
            Local Name: tr
            Node Value: null
            Namespace URI: http://www.w3.org/1999/xh
            Attribute 0 (class)
            Type: ELEMENT_NODE
            Child 0 (TH)
                Node Name: TH
                Local Name: th
                Node Value: null
                Namespace URI: http://www.w3.org/1999
                Attribute 0 (rowspan)
                Type: ELEMENT_NODE
                Child 0 (#text)
                    Node Name: #text
                    Node Value: Train Number
                    Type: TEXT_NODE
```

Fig 3 source code and corresponding DOM tree Fragment

As shown in fig 1 section selection consist of three steps;

Step 1: Identifying data section or region of interest on page

This step is performed with the help of interactive interface. This interface helps user to identify sections in web page including table region, paragraph region, and list region etc. Output from this step, is that we obtain extraction pattern for that particular region. The detail process to construct pattern is given in next section. Fig 4 shows selected section, selected section is highlighted with gray color.

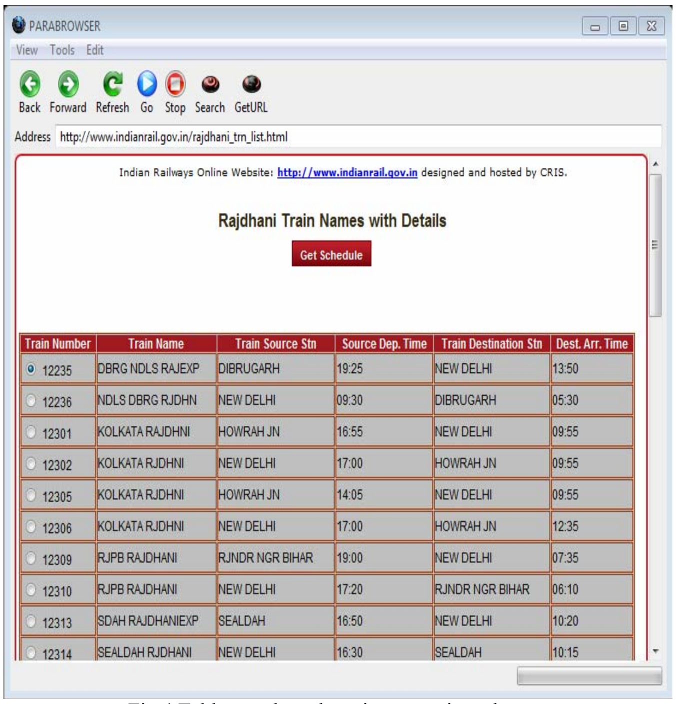
Step 2: Identifying important semantic tokens

In this step, Process collects semantic tokens, which permits extractor to walk through DOM tree and highlight semantic tokens of interest in source page.

Step 3: Identifying useful hierarchical structure.

In this step, hierarchical parent structure of selected element is retrieved. The output of this step is set of semantic tokens, which can be used to resolve ambiguity of pattern (if any).

In summary, section selector analyses DOM tree and formatting information of web page to extract semantic token of interest. In next section we will see, how to construct pattern and how to extract data with that pattern.


PARABROWSER
View Tools Edit
Back Forward Refresh Go Stop Search GetURL
Address http://www.indianrail.gov.in/rajdhani_trn_list.html
Indian Railways Online Website: http://www.indianrail.gov.in designed and hosted by CRIS.
Rajdhani Train Names with Details
Get Schedule

| Train Number | Train Name | Train Source Stn | Source Dep. Time | Train Destination Stn | Dest. Arr. Time |
|---|---|---|---|---|---|
| ● 12235 | DBRG NDLS RAJEXP | DIBRUGARH | 19:25 | NEW DELHI | 13:50 |
| ○ 12236 | NDLS DBRG RJDHN | NEW DELHI | 09:30 | DIBRUGARH | 05:30 |
| ○ 12301 | KOLKATA RAJDHNI | HOWRAH JN | 16:55 | NEW DELHI | 09:55 |
| ○ 12302 | KOLKATA RJDHNI | NEW DELHI | 17:00 | HOWRAH JN | 09:55 |
| ○ 12305 | KOLKATA RJDHNI | HOWRAH JN | 14:05 | NEW DELHI | 09:55 |
| ○ 12306 | KOLKATA RJDHNI | NEW DELHI | 17:00 | HOWRAH JN | 12:35 |
| ○ 12309 | RJPB RAJDHANI | RJNDR NGR BIHAR | 19:00 | NEW DELHI | 07:35 |
| ○ 12310 | RJPB RAJDHANI | NEW DELHI | 17:20 | RJNDR NGR BIHAR | 06:10 |
| ○ 12313 | SDAH RAJDHANIEXP | SEALDAH | 16:50 | NEW DELHI | 10:20 |
| ○ 12314 | SEALDAH RJDHANI | NEW DELHI | 16:30 | SEALDAH | 10:15 |

Fig 4 Table as selected section to retrieve data

## V. Pattern Constructor And Data Extractor

This section focuses on constructing pattern. Before seeing details of algorithm to construct pattern, we will see some basics of CSS selector and information of DOM tree nodes. DOM tree nodes are classified in about 12 types as attr, element, text, cdatasection, entity reference, entity, comment, document, document type, processing instruction, document fragment, and notation. Out of which we are considering attr node only to extract data. Node type of attribute node is attr, node name returns attribute name, and node value returns attribute value. We are using attribute nodes as a CSS selector, to construct pattern. There are various types of CSS selectors like Universal selector, attribute selectors, descendant selectors, type selectors, child selectors, adjacent sibling selectors, id selectors, and class selector.

### *A. Attribute Selector*

Here we will first see attribute selector in detail. Attribute selector allows us to specify rules that match elements which have certain attributes defined in the source document. Attribute selector's matches in following four ways:

i. [att]
In this type when the element sets the "att" attribute, whatever the value of the attribute is matched.

ii. [att=val]
In this type when the element has "att" attribute value is exactly "val" is matched.

iii. [att~=val]
This type represents an element with the "att" attribute whose value is a white space-separated list of words, one of which is exactly "val". If "val" contains white space, it will never represent anything. If "val" is the null string, it will not represent anything.

iv. [att|=val]
This type represents an element with the “att” attribute, its value either being exactly “val” or beginning with “val” immediately followed by “-”. This is to permit language sub code matches. E.g. the following rule will match for values of the “lang” attribute that begin with “en”, including “en”, “en-US”, and “en-cockney”: *[lang|="en"];

## B. *Class Selectors*

Along with attribute selectors we can also use class selector. We can use the period (.) notation as an alternative to the ~= notation for representing the class attribute. Thus, div.value and div[class~=value] have the same meaning. The attribute value must be immediately preceded by “period” (.). e.g. “.intro”: Selects all elements with class=”intro”.

## C. *Id Selectors*

Document may contain attributes that are declared of type ID. Attributes of type ID is special because two attributes never have same value. An ID attribute can be used to uniquely identify its element in document. In HTML all ID attributes are named by “id”. The ID attribute of a document allows us to assign an identifier to one element instance in the document tree. An element is matched using CSS ID selectors based on its identifier. A CSS ID selector contains a “#” immediately followed by the ID value, which must be an identifier. e.g. “#firstname” - Selects the element with id=”firstname”

Along with attribute, class and id selector other selectors like universal selector (*), child selector (>), adjacent selector (+) etc. are used to construct pattern. As shown in fig 4 the web page of Indian rail which contain the information about rajdhani train names with details, which is rendered by our web page renderer. Rendered web page contains all the data about the train in table, this table is highlighted in gray. The small part of DOM tree for document rajdhani train details is shown in fig 5. From fig 5 we can see that table contains different attributes like width, cellspacing, cellpading, border, and class. These attributes are shown by dashed lines in figure.

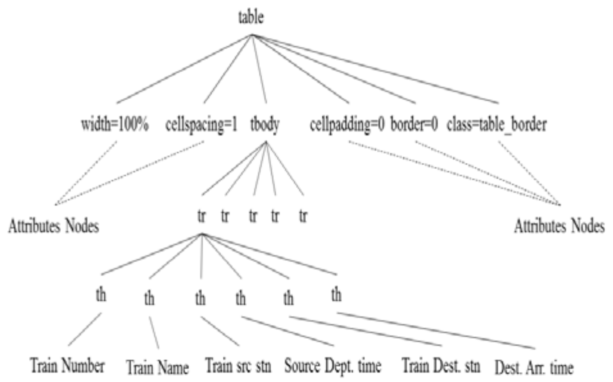


Fig 5 DOM Tree Fragment of rajdhani train details page

As shown in fig 4 when we select the table here we called section containing details of web page, rajdhani train names is highlighted with gray color, and simultaneously all the attributes of table are retrieved. For retrieving attributes we used DOM tree.

Let’s assume that given input page “p”, contain data in specific section, this section is nothing but the some tag or element of DOM tree. “l” is the list to store the attribute names and “av” is the list to store attribute values. Input to algorithm is input web page and selected element. Algorithm AttributeRetrival gives details of how we are extracting attribute from web page.

```
Algorithm AttributeRetrieval (p,t)
// p is the given input page
// t is the selected element
// a is current retrieved attribute
// n is name of retrieved attribute
// v is value of retrieved attribute
1. Initialize l,av; i=0;
2.    for each attr a of selected element t
3.        retrieve name n of a;
4.        retrieve value v of a;
5.        l[i]=n;
6.        av[i++]=v;
7.    end for
8.  return l, av;
```

Algo 1 Algorithm to retrieve the attributes

Once we have all attributes of selected element, we can now construct the pattern. Here, we will see the algorithm to construct the pattern. As already stated, data region containing data is also sub tree of DOM tree. All the data to be extracted is under one element of DOM tree. Data is usually contained under block level tag such as table, list etc. for constructing pattern we need retrieve name of selected element from DOM tree.

Here we are taking queue data structure “p” to construct pattern. Two lists “l” and “av” which are retrieved from AttributeRetrieval algorithm are given as input to the algorithm. In queue “q” we first insert the retrieved element name, then we insert all the attributes in the form (a, v). In this way we constructing pattern to extract data. Details of algorithm are given below.

From retrieved attributes by algorithm 1 pattern is constructed by algorithm 2. Below is the pattern for selected table:
***table[width=100%][cellspacing=1][cellpadding=0][border=0][class=table_border].***

```
Algorithm ConstructPattern (p, t, l, av)
// p is the given input page
// t is the selected element
// l is list containing name of attributes
// av is list containing values of attributes.
1. Initialize p; i=0;j=0;
2. n=sizeOf(l);
3. p[j++]=insert (t.name);
4.    for each i from 0 to n
5.          p[j++]= insert ([);
6.          P[j++]= insert (l[i]);
7.          p[j++]= insert (]);
8.          p[j++]= insert (=);
9.          p[j++]= insert ([);
10.         P[j++]= insert (av[i]);
11.         p[j++]= insert (]);
12.   end for
13. return p;
```

Algo 2:Algorithm to construct pattern.

### *D. Data Extractor*

In this section we will see, how data is retrieved with help of pattern generated in previous section. Here we derive two cases to extract data. We are making two cases based on uniqueness of identifying data section from pattern. Before deriving two cases we will first see the algorithm to extract data. Algorithm is given below.

```
Algorithm ExtractData (q , T, l)
// l is list containing attributes
// q is the pattern derived by algo 2
// {a,v} is the attribute value pair.
// T is the DOM tree for page.
1. Flag=false;n=0;
2. retrieve element e from q
3. match e in T
4. if match found then
5.    n= sizeOf(l);
6.    For each i from 0 to n
7.        retrieve {a,v} pair
8.        match {a,v} in T in same sub tree where
          e is matched
9.     endfor
10. endif
11. retrieve text from matched sub tree
12. return text
```

Algo 3 Algorithm to retrieve the attributes

Input to algorithm is "q" data structure containing pattern, tag tree, and list "l". In above algorithm we are first matching the selected element in tree. When match is found, we are matching the attributes. When all attributes and element is matched we are retrieving all the data within that element. Now, we will see two cases to extract data.

**Case 1:** Pattern identifies two or more data section.

In this case there is ambiguity in identifying the correct section with given pattern, e.g. consider the page containing more than one data record. Let's say page contains two tables, one is under div tag another is under td tag, and both of these tables have the same number of attributes, and also the values of all attributes are same. so pattern constructed by algorithm 2 fails to identify the correct data section uniquely. Consider the fig 6 formulating the problem.

As shown in fig 6 both tables have same number of attribute and also, name and value of each attribute is same. So when we start retrieving data, data from both sections is retrieved. Because, in data retrieval module we are first matching element name in DOM tree, when element is found then we are matching the attribute of pattern with the attribute of element in DOM tree. Here pattern matches two time in document so there is ambiguity in retrieving data. Here we need the pattern validator.

### *E. Pattern Validator*

When there is ambiguity in retrieving data, we need to validate pattern, and we need to change the pattern unless user gets the intended data. To solve the problem, we are adding parent element to pattern, this process continues still we get the unique pattern which will give us only required section. When we add parent element to our pattern then we obtains unique patterns which will point to intended section. E.g.

*td>table[ width=100%][cellspacing=1][cellpadding=0][border=0][class=table_border].*

Parent element of another table is „div" so ambiguity is resolved. Now, we can extract data in usual way by using our algorithm.

**Case 2:** Pattern uniquely identifies the correct data section.

In this case we can directly use pattern to extract data, as there is no issue of ambiguity. Here directly element is matched in DOM tree, when element is matched then all attributes from pattern are matched with specific sub DOM tree. Text values from this sub tree are retrieved. In this way, we get all required data from web page.

Let's see the one example to extract data with given pattern. Assume that input page to extract data is page shown in fig 4. We want to extract data from selected table. Selected table is highlighted with gray color. Pattern to extract data from this table is:

***table[width=100%][cellspacing=1][cellpadding=0][border=0][class=table_border].***

We designed pattern in relative type, so that we don't need to start from root of DOM tree. If pattern is absolute we need to start from root node, and every time, we need to follow same path from root to that node. So if some element is added in page or page structure changes then absolute pattern fails.

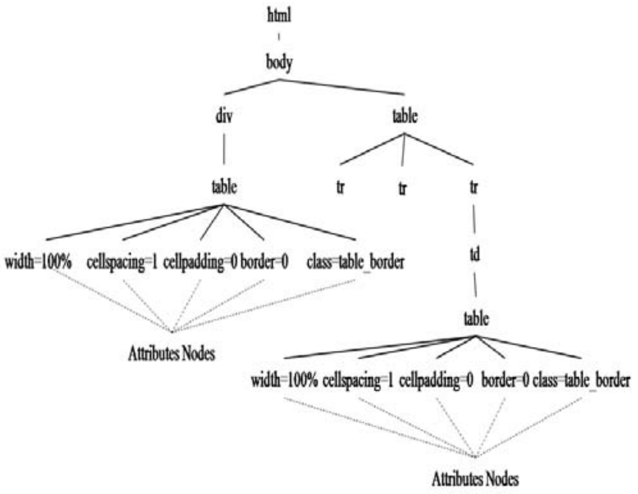


Fig 6: DOM tree of page showing ambiguity in indentifying section

Our pattern is relative, so we start with matching selected element in DOM tree, in above pattern selected element is table, so table node is matched in DOM tree. But there may be one or more table nodes in DOM tree, so we need use remaining attribute from pattern to match specific table element. After table element is matched we start to match attribute named width in DOM tree. When this attribute is matched exactly, we match remaining attributes in same way. When whole pattern is matched, we start retrieving the text values from matched sub tree.

## VI. Experiments

This section evaluates our system, DEUDS (Data Extraction Using DOM Tree and Selector), which implements the above discussed algorithms. The evaluation of system consists of two parts:

1. Data record section identification: In this step we are verifying that, proposed system correctly identifies the data record section or not.
2. Data record extraction: In this step, we are verifying that proposed system correctly extracts data or not. Use no additional space above the subsection head.

### A. Setup

Experiments are performed on a machine equipped with an Intel Core 2 Duo processor working at 2.40 GHz clock speed and 1066 MHz FSB, with 3 GB of RAM. Operating system Windows 7 and Java Development Kit 1.7 are used for experiment.

Experiments are performed on 290 different input pages, collected from different web sites. These pages were obtained as follows:

- RoadRunner [5]: (19 collections) These are all the collections available at RoadRunner site [22].
- RISE [23]: (6 collections) Distributed repository called RISE, consist of online information sources that are used for the empirical analysis of learning algorithms that generate extraction patterns. Only 6 out of the 10 collections from RISE are used in our extraction problem.
- The rest web pages are crawled by us from various well-known sites like indianrail, bookboon, flipkart, dejavutrends.com and shopping.yahoo.com etc.

### B. Evaluation

Table 1 shows performance of our proposed technique. SI indicates section identification, DE indicates data extracted. An cr and pcr means correct and partially correct respectively, whereas wr means wrong. Column 1 of table 1 contains the information about the page source i.e. URL of page. Column 2 indicates description of pages. Column 3 indicates number of pages from each source. Column 4 and 5 indicates number of pages from which data section is correctly identified and number of pages from which data section is partially correct or wrongly identified. Column 6 and 7 indicates number of pages from which data section is correctly retrieved and number of pages from which data section is partially correct or wrongly retrieved. For measuring performance of our system, we used 290 pages. Out of 290 pages, data section of 280 pages is correctly identified; hence data extracted from 280 pages is also correct. For 10 pages DEUDS failed to identify data sections correctly, so data retrieved from these 10 pages is partially correct. For measuring performance we considered partial correct data as wrong data. After evaluation we got values of precision and recall as 96.55%. From experiments, we got conclusion that, our system is 100% efficient for web pages containing data in table format or list format.

We are comparing our system with RoadRunner [5]. Experiment shows that our system is much efficient than RoadRunner in detecting sections and extracting data. We used same input pages, as those are used for evaluating RoadRunner. RoadRunner failed to extract correct data from about 21 pages. Data retrieved by RoadRunner is partial from collection “national team info” from uefo.com. So performance of our system, DEUDS is much better than Road Runner.

## VII. Related Work

Lot of recent work is there related to Information Extraction, classified along different parameters: targeted information sources, automation degree, and schema identification. Section 1 explained some of the related work. Here we will see the differences between our work and, RoadRunner.

Our work is related to the ROADRUNNER [5]. Model of page creation is used in RoadRunner, that is similar to ours data section identification. RoadRunner assumes site generation is process of encoding the database content into HTML code. As a consequence, data extraction is considered as a decoding process. As a result, generating a wrapper for a set of HTML pages corresponds to inferring a grammar for the HTML code. RoadRunner begins with the input page as its initial template. Then, It checks for each subsequent pages that page can be generated by the current template. If it cannot be, it modifies its current template. There are several limitations to the RoadRunner approach:

1. RoadRunner assumes that every HTML input pages is generated by the template. This assumption is crucial in RoadRunner to check template of each page. For many web-sites pages, this assumption is clearly wrong since html tags can also be added in between data values.
2. RoadRunner might fail to produce any output if there is change in the input template.
3. Complicated operations are performed to search a new template; When RoadRunner finds that the current template does not generate an input page.

## VIII. Conclusions

In this paper, we proposed a new approach for constructing pattern to extract data, called DEUDS to solve problem of page-level data extraction. Our proposed approach works in three stages web page renderer, Section selector, and Pattern generator. We have already seen the details of each stage. In earlier work, extraction rules are learned from training examples. In this paper, we presented an unsupervised approach to pattern discovery.

Many approaches have been proposed by different researchrs for Web data extraction ([21],[1] represents survey on web data extraction tool),but few of them works (RoadRunner, EXALG and FiVaTech) solve this problem at a page level. Proposed technique gives very good result in extracting data, and also pattern generated by our technique is stable. Pattern generated by our technique is stable because, we are generating pattern in relative manner not in absolute manner. In future work, we plan to extend our approach to extract hidden data, and to automatic label extracted data.

TABLE I Performance measure of DEUDS system

| **Page information** | | | **SI** | | **DE** | |
|---|---|---|---|---|---|---|
| **Page src** | **Page info** | **Num** | **cr** | **wr/pcr** | **cr** | **wr/pcr** |
| amazon.com | pop artist by style | 19 | 19 | 0 | 19 | 0 |
| amazon.com | cars by brand | 21 | 21 | 0 | 21 | 0 |
| buy.com | product subcategories | 20 | 20 | 0 | 20 | 0 |
| buy.com | product information | 10 | 0 | 10 | 0 | 10 |
| rpmfind.net | packages by distribution | 20 | 20 | 0 | 20 | 0 |
| rpmfind.net | packages by maintainer | 20 | 20 | 0 | 20 | 0 |
| uefa.com | players in the national team | 20 | 20 | 0 | 20 | 0 |
| uefa.com | national team info | 20 | 20 | 0 | 20 | 0 |
| wine.com | accessories | 11 | 11 | 0 | 11 | 0 |
| wine.com | wines by producers | 10 | 10 | 0 | 10 | 0 |
| majorleguebaseball.com | players by initial | 10 | 10 | 0 | 10 | 0 |
| majorleguebaseball.com | player statistics | 10 | 10 | 0 | 10 | 0 |
| nba.com | team stats | 10 | 10 | 0 | 10 | 0 |
| nba.com | team roaster | 10 | 10 | 0 | 10 | 0 |
| RISE | LA Restaurants | 28 | 28 | 0 | 28 | 0 |
| RISE | Pharma Web | 10 | 10 | 0 | 10 | 0 |
| RISE | Corel | 21 | 21 | 0 | 21 | 0 |
| Indianrail.com | Indian rail | 4 | 4 | 0 | 4 | 0 |
| Flipkart.com | Eng. books | 1 | 1 | 0 | 1 | 0 |
| Dejavutrands.com | Computers and laptops | 5 | 5 | 0 | 5 | 0 |
| shopping.yahoo.com | laptop | 10 | 10 | 0 | 10 | 0 |
| | total | 290 | 280 | 10 | 280 | 5 |
| | Precision | 280/290 | 96.55% | Recall | 280/290 | 96.55% |

## REFERENCES

[1] C.-H. Chang, M. Kayed, M.R. Girgis, and K.A. Shaalan, "Survey of Web Information Extraction Systems," IEEE trans. Knowledge and Data Eng., vol. 18, no. 10, pp. 1411-1428, Oct. 2006.

[2] C-N. Hsu and M-T. Dung, "Generating finite-state transducers for semi-structured data extraction from the Web Information Systems," 23(8):pp. 521–538, 1998.

[3] N. Kushmerick, D. Weld, and R. Doorenbos, "Wrapper induction for information extraction," In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI), 1997.Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd.

[4] I. Muslea, S. Minton, and C. Knoblock, "A hierarchical approach to wrapper induction," In Proceedings of the 3rd International Conference on Autonomous Agents (Agents "99), Seattle, WA, 1999.

[5] V. Crescenzi, G. Mecca, and P. Merialdo, " RoadRunner: towards automatic data extraction from large Web sites," Proceedings of the 26th International Conference on Very Large Database Systems (VLDB), Rome, Italy, 2001, pp. 109-118.

[6] Y. Zhai and B. Liu, "Web Data Extraction Based on Partial Tree Alignment," Proc. Int"l Conf. World Wide Web (WWW-14), 2005, pp. 76-85.

[7] J. Wang and F.H. Lochovsky, "Data Extraction and Label Assignment for Web Databases," Proc. Int"l Conf. World Wide Web (WWW-12), 2003, pp. 187-196.

[8] A. Arasu and H. Garcia-Molina. "Extracting structured data from web pages," In SIGMOD Conference, New York, NY, USA, 2003, pp. 337–348.

[9] C.-H. Chang and S.-C. Lui, "IEPAD: Information Extraction Based on Pattern Discovery," Proc. Int"l Conf. World Wide Web (WWW-10), 2001, pp. 223-231.

[10] B. Liu, R. Grossman, and Y. Zhai. "Mining data records in web pages," In SIGKDD conference, New York, NY, USA, 2003, pp. 601–606.

[11] H. Zhao, W. Meng, and C. Yu. "Automatic extraction of dynamic record sections from search engine result pages," In VLDB Conference, 2006, pp. 989–1000.

[12] H. Zhao, W. Meng, and C. Yu. "Mining templates from search result records of search engines," In SIGKDD Conference, New York, NY, USA, 2007, pp. 884–893.

[13] A. Sahuguet and F. Azavant , "Building intelligent Web applications using lightweight wrappers," Data and Knowledge Engineering 36(3): 283-316, 2001.

[14] L. Liu , C. Pu, and W. Han, "XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources," Proceedings of the 16th IEEE International Conference on Data Engineering (ICDE), San Diego, California, 2000, pp. 611-621.

[15] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. "Fully automatic wrapper generation for search engines," In WWW Conference, New York, NY, USA, 2005, pp. 66–75.

[16] K. Simon and G. Lausen , "Viper: augmenting automatic information extraction with visual perceptions," In CIKM Conference, pages 381–388, New York, NY, USA, 2005.

[17] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser. "Extracting data records from the web using tag path clustering," In WWW Conference, 2008, pp. 981–990.

[18] W. Liu, X. Meng, and W. Meng, "Vide: A vision-based approach for deep web data extraction," IEEE Transactionson Knowledge and Data Engineering, 22:447–460, 2010.

[19] S. Chakrabarti, "Mining the Web: Discovering Knowledge from Hypertext Data," Morgan Kaufmann Publishers, 2002.

[20] Mohammed Kayed and Chia-Hui Chang, "FiVaTech: Page-Level Web Data Extraction from Template Pages," IEEE transactions on knowledge and data engineering, vol. 22, no. 2 , pp. 249-263, February 2010.

[21] A.H.F. Laender, B.A. Ribeiro-Neto, A.S. Silva, and J.S. Teixeira, "A Brief Survey of Web Data Extraction Tools," SIGMOD Record, vol. 31, no. 2, pp. 84-93, 2002.

[22] http://www.dia.uniroma3.it/db/roadRunner/experiments.html.

[23] http://www.isi.edu/info-agents/RISE/repository.html.