CHAPTER-7

WEB SCHEMA DETECTION AND DATA EXTRACTION SYSTEM

7.1 Introduction:

For web data applications, web data abstraction is the significant component. It is difficult task to extract data from deep web pages. Several websites contain huge sets of pages generated using regular template or layout. For example, Amazon site includes the author, title, comments, etc. in the similar manner in all of its book pages. The values used to generate pages typically come from a database. So, there is a need to extract the values from templates generated web pages automatically.

The ultimate goal of the proposed module is to provide unsupervised page-level data extraction approach to deduce matching schema for template pages. In this module, web pages are compared based on visual clues and data region from web pages are extracted. If they belong to a fixed template, then we detect schema by applying tree merging, tree alignment, and mining techniques. The experiments show a good result for the web pages used in many web data extraction. Also, time-based comparison is done for data extraction by removing noise blocks from web pages and without removing noise blocks.

7.2 Problem definition:

Develop a subsystem that will detect schema and extract data from the template generated web pages automatically.

7.3 Objective:

Objectives of proposed modules are

1. To detect schema and extract data automatically from template generated pages.
2. To develop efficient method to extract data from web pages by removing noise blocks.

7.4 Development of proposed module:

A main feature of pages belonging to the same website pages shares the same template, because they are encoded in a regular manner across all pages. For example, business websites regularly have a template for displaying company logos, browsing menus and copyright declarations, such that all pages of the same website look consistent and designed. Also, templates can be used to provide a list of report to show objects of the same type. From template pages, extraction of information can be applied in several circumstances.

7.4.1 Layout of proposed module:

Proposed system is used to detect matching schema and deduce template from template generated pages. Automatic data extraction can be done by identifying schema. Features of Proposed Approach:

- It is page-level web data extraction task.
- It is an unsupervised approach that does not require training sets or manually generated rules.

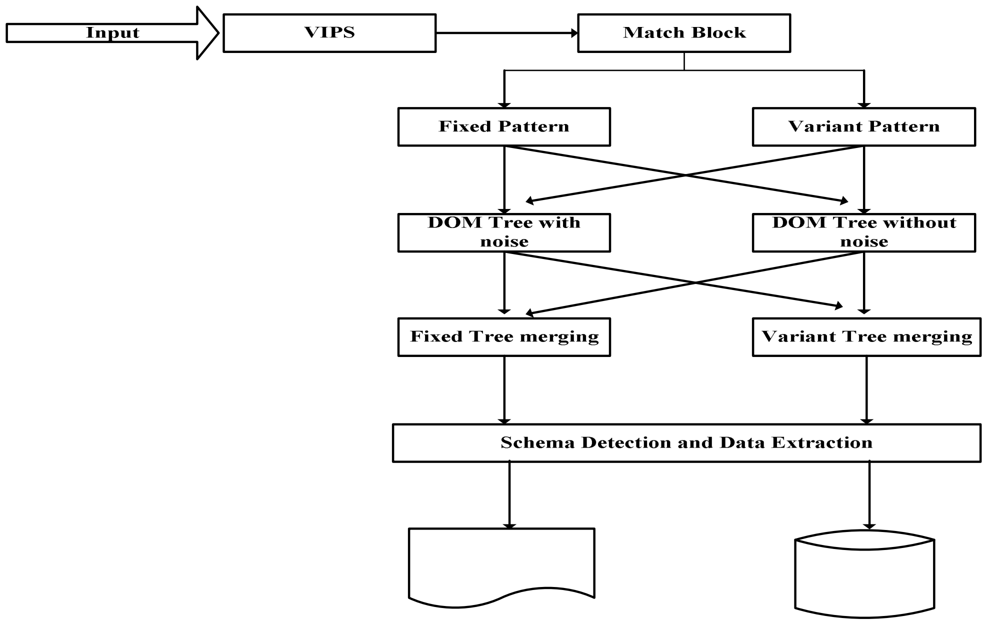


Figure 7.1 Proposed Framework for schema and data extraction

Proposed Work:

Detecting the schema of a web site has been a key step for value-added services like comparative shopping and information integration systems.

Our proposed work is aimed to filter schema of website and extract data.

➢ **STEPS :**

Step 1: Take two web pages as input.

Step 2: For each page, we apply Vision-based Page Segmentation (VIPS) algorithm to Segment Web page and to build visual block tree.

Step 3: Fixed/variant template web pages are detected by comparing blocks inVisual block trees..

Step 4: Noise blocks are removed from DOM trees.

Step 5: For fixed template pages, we apply multiple tree merging algorithm [1],

This consists of following steps.

- Identification of peer nodes
- Alignment of matrix
- Repetitive pattern mining
- Merging of optional nodes

Step 6: Pattern tree is created and schema is detected.

Step 7: Data extraction is done by matching pattern tree and html tree.

Step 8: From variant template pages, data is extracted.

7.5 Algorithm used for Web Schema Detection and Data Extraction System:

7.5.1 Page Segmentation Using Vips Algorithm:

For each input page, VIPS (VIsion-based Page Segmentation) algorithm[31] is used to create visual block tree. By using VIPS (VIsion-based Page Segmentation) algorithm web page is segmented into different blocks. To perform segmentation, VIPS considers both the page appearance and Document Object Model (DOM).

There are three main steps in VIPS that are block extraction, separator detection and content structure construction. These three steps as a whole are regarded as a round. The approach is top-down. The web page is firstly segmented into several big blocks and the hierarchical structure of this level is recorded. For each big block, the same segmentation process is carried out recursively until a user defined threshold is reached. The threshold is called Permitted Degree of Coherence (PDoC) and this is based on Degree of Coherence (DoC). DoC is a numeric value between 1 and 10 that increases as consistency between blocks increases.

- **Steps in VIPS**
    - Visual Block Extraction

- Visual Separator Detection
- Content Structure Construction

7.5.2 Fixed/Variant Template Detection Based On Visual Block Tree:

In this, using VIPS algorithm each web page is transformed into visual block tree and visual features are extracted. Then visual blocks in two trees are compared based on visual features like background color. If percentage of matched blocks is greater than or equal to threshold (= 60% in our experiment), then pages belong to fixed template, otherwise they belong to variant templates.

Algorithm 1:Fixed/Variant Template Detection(VT1,VT2)[31]

Input : Visual Block trees VT1 and VT2 of web pages

Output : Fixed/Variant template

```
1.  Initialize threshold=60%;
2.  Compare blocks in VT1 and VT2 based on visual features
3.  If percentage of matching blocks >= threshold
4.  Fixed Template Page
5.  else
6.  Variant Template Page
7.  EndIf
```

7.5.3 Noise Block Removal:

In this, DOM trees are filtered out by removing noise blocks. The Web usually contains two types of blocks in the generated Web pages : template data blocks and data rich blocks. Template data blocks contains irrelevant template data such as advertisements, navigational panels. Data rich blocks contains relevant data in which user is interested. Removing template data blocks from the pagesbefore the extraction process is a very important step as it improves the efficiency ofthe extraction algorithm and helps us to get accurate results. In this, an algorithm is proposed to remove irrelevant template blocks before applying the peer nodes recognition step for data extraction.

Algorithm Noise_Block _Removal (TreeNode)[31]

Input : TreeNode

Output :DOM tree by removing noise blocks

```
1. Find area of each child  in Treenode
2. if (the area of each child in TreeNode< 40%) return NULL;
3. biggest = the child with biggest percentage area;
4. for each child in TreeNode
5. If (child != biggest) then
6. Remove images from corresponding to tag child
7. Endif
8. Noise_Block_Removal(biggest)
```

7.5.4 Tree Merging Algorithm :

In this, all input DOM trees are merged into a structure called fixed/variant pattern tree, whichis used to detect the schema of the Website.

It consists of the four steps, identification of peer nodes, and alignment of matrix, pattern mining, and optional node detection .

Algorithm DOMTreeMerge (T,P)[31]

Input:

i. T is set of DOM trees of same type.

ii. P is tag for the roots of T.

Output : Pattern Tree

```
1.  Based on T form a matrix M where each tree t in T is presented in a one column;
2.  Find Peer Nodes in M;
3.  Obtain childList by alignment of matrix M ;
4.  Apply patternMining on childList to detect repetitive patterns starting with length 1;
5.  Apply optional node merging on childList;
6.  For each node n in childList
7.  If ( n is a tree ) then
8.  C= DOMTreeMerge ( peerNode (n, M), tag (n))
9.  else  //n is leaf node
10. C=n;
11. Endif
12. Insert C as child of P;
13. EndFor
```

```
14.    Return pattern tree ;
```

➢ **Identification of peer nodes**

In the peer node recognition step, 2-trees matching algorithm is used. The algorithm returns a matching score which is a normalized ratio between the numbers of pairs in the mapping over the maximum size of the two trees. The two trees are considered as peer trees, if the score is greater than a threshold (0.5).By testing on different web pages , the threshold is determined as 0.5 as if matching score is greater than or equal to 0.5, the trees are matched.

7.5.5 Tree Matching Algorithm for fixed template pages

Algorithm TreeMatchScore (T1,T2)[31]

Input: T1 and T2 are two trees.

Output : Matching Score

```
1.     If  T1. text! = T2.text
2.     Return 0;
3.     If (T1 or T2 is leaf) or size (T1) ==size (T2)) then
4.     Return 2* TreeMatching (T1, T2)/ size (T1) + size (T2);
5.     Score=0.0; k=no of children of T1
6.     For each child c in T1 do
7.     nodeScore=0.0; matchno=0;
8.     For each child d in T2 do
9.     tmp=2*TreeMatching (c, d)/size (c) + size (d);
10.    if (tmp>θ)
11.    nodeScore+=tmp; matchno++;
12.    Endif
13.    EndFor
14.    If (matchNo>0) nodeScore=nodescore/matchno;
15.    Score+=nodeScore;
16.    Endfor
17.    Return (score/k+2/(size(c)+size(d)) ;
```

7.5.6 Tree Matching Algorithm for variant template pages:

Alignment of matrix:

In this, matrix M is transformed into associated matrix by aligning nodes in peer matrix. In this, matrix is checked row by row. If every column in row contains same symbol or it is text node, the row is considered as aligned row.

If row is not aligned in matrix, then node is shifted depending on span value of node. Span value of node is highest number of diverse nodes between any two successive occurrences of node in each column plus one. Following rules are used in order to determine column to be shifted from current row r and for identifying required shift distance.

Algorithm MatrixAlignment (M)[31]

Input : Peer Matrix M

Output : List of aligned nodes

```
1.  r=0;
2.  shiftlen=0;
3.  while (M is not aligned) do
4.  while (row r in M is not aligned) do
5.  Select column to be shifted from current row r and shift column by shiftlen
    distance by applying given rules;
6.  End while
7.  r++;
8.  End while
9.  childList=getAlignmentResult(M);
```

The node to be shifted if M is not aligned depends on span value of node. Span value of node is defined as the maximum number of different nodes between any two consecutive occurrences of node in each column plus one. The column to be shifted is selected from current row r and identifies the required shifted distance (shiftLen) by applying following rules in order:

Rules for Matrix Alignment:

1. Rule R1 : A column c is selected, from left to right, such that there exists a node with same symbol at upper row $r_{up}$where $M[r_{up}][c']=n$ for some c' and $r-r_{up}< span(n)$.Then column c is shifted with distance equal to 1.
2. Rule R2: If R1 fails, then select column c with nearest row r_{down} from r such that $M[r_{down}][c']=M[r][c]$ for some $c\neq c'$. In such case, column is shifted with distance equal to $r_{down}-r$.

3. Rule R3: if both rules R1 and R2 fail, then check if r contains all data (text/image) nodes. In this, no shifting is done.
4. If all of R1, R2 and R3 fail, we select the symbol that occurs the maximum number of times on this row. Keeping all columns with that symbol in r unchanged; shift all other columns down by one.

➢ **Pattern Mining:**

To find repetitive patterns in the input and to discover set-type data this step is performed. However, there can be many repetitive patterns discovered and a pattern can be embedded in another pattern, so it makes difficult to the detect schema. In this, we detect every consecutive repetitive pattern (tandem repeat) and merge them (by deleting all occurrences except for the first one) from small length to large length.

➢ **Optional Node Merging:**

After the mining step, optional nodes are detected based on the occurrence vectors. The occurrence vector of a node c is defined as the vector (b_1, b_2,. . . ,b_u), where b_i is 1 if c occurs in the i^{th} occurrence, or 0 otherwise. If c is not part of a set type, u will be the number of input pages. If c is part of a set type, then u will be the summation of repeats in all pages. With the occurrence vector defined above, optional nodes are grouped based on the following rules and add to the pattern tree one virtual node for the group.

Rule 1:

If a set of adjacent optional nodes have the same occurrence vectors, then they are grouped as optional.

Rule 2:

If a set of adjacent optional nodes have complement occurrence vectors, then they are grouped as disjunction.

Rule 3:

If an optional node is a fixed node, then it is grouped with the nearest non fixed node
A virtual node is added for each merged optional and disjunctive just like set-type data.

7.5.7 Schema Detection:

Detecting structure of website includes identifying schema of website. The task for schema is to recognize basic type, set type, optional type and tuple type. Here, the system traverses fixed-variant pattern tree from root downward and marks nodes as k-order or k-tuple. For nodes with only one child and not marked as set or optional type, there is no need to mark it as 1-tuple.

7.5.8 Data Extraction:

Once the Pattern Tree has been constructed, the tree is traversed from top to bottom in both the HTML trees and the Pattern Trees simultaneously, matching both at each level and then descending down recursively. The algorithm returns all matched variant nodes and extracts data.

a. UML Design Modeling:

i. Schema Detection And Data Extraction Module:

a) Class Diagram:

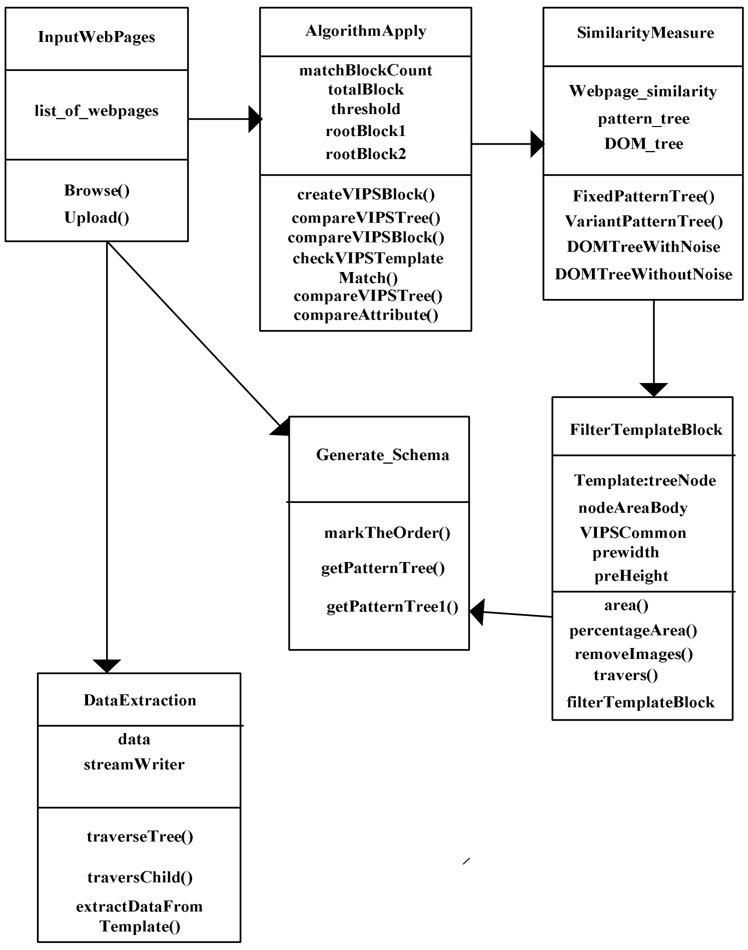


Figure 7.2 Class diagram for schema detection and data extraction

In fig 7.2 the system classes have been defined namely InputWebPages, Algorithm Apply, Similarity Measure, FilterTemplateBlock, Generate_Schema andDataExtraction. Different attributes and methods are useful for defining the class.

b) Sequence Diagram:

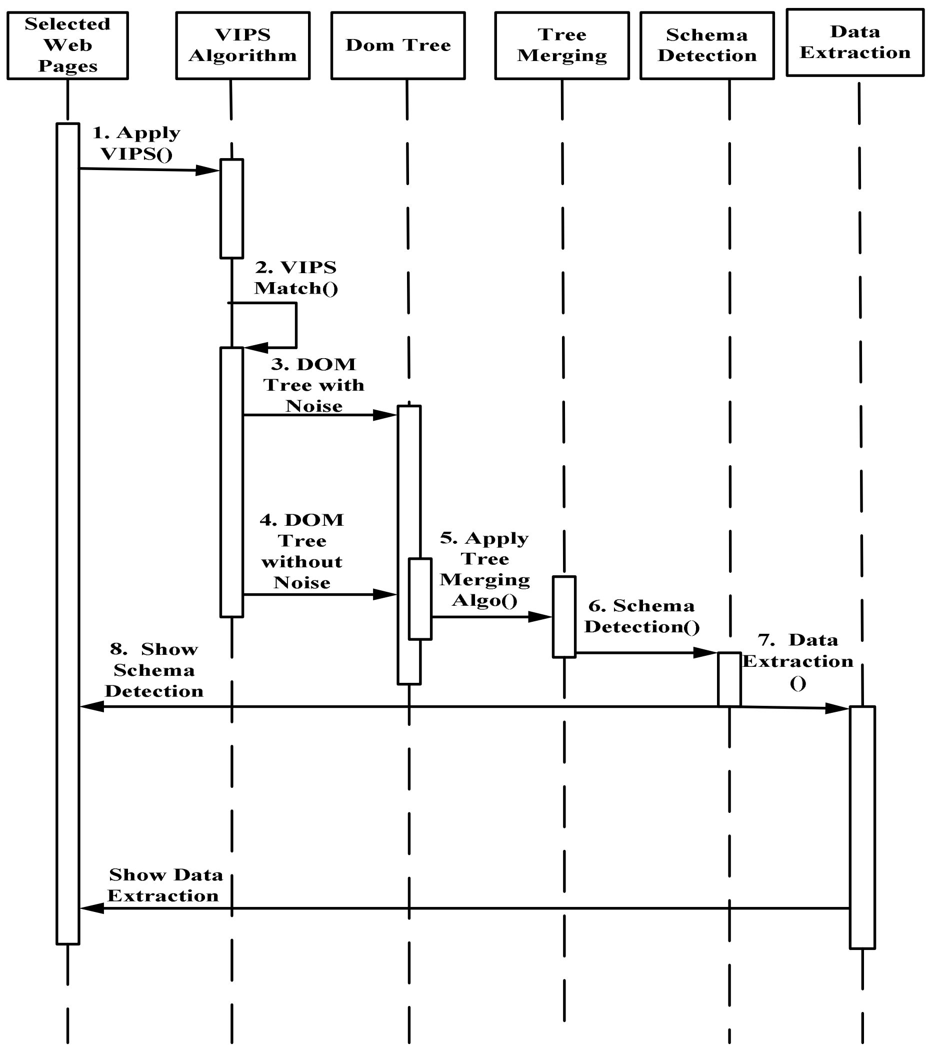


Figure 7.3 Sequence diagram for schema detection and data extraction

In fig 7.3 we have used different objects namely Selected Web Pages, VIPS Algorithm, DOM tree, Tree merging, schema detection and data extraction These objects interacts with each other and also shows lifeline of each object. It is also one type of interaction diagram. Sequence diagrams are useful in documenting how a future system should behave. Sequence diagram consists of lifelines. Lifelines represent either roles or objects instances which are participated in the sequence being modeled.

c) Use Case Diagram:

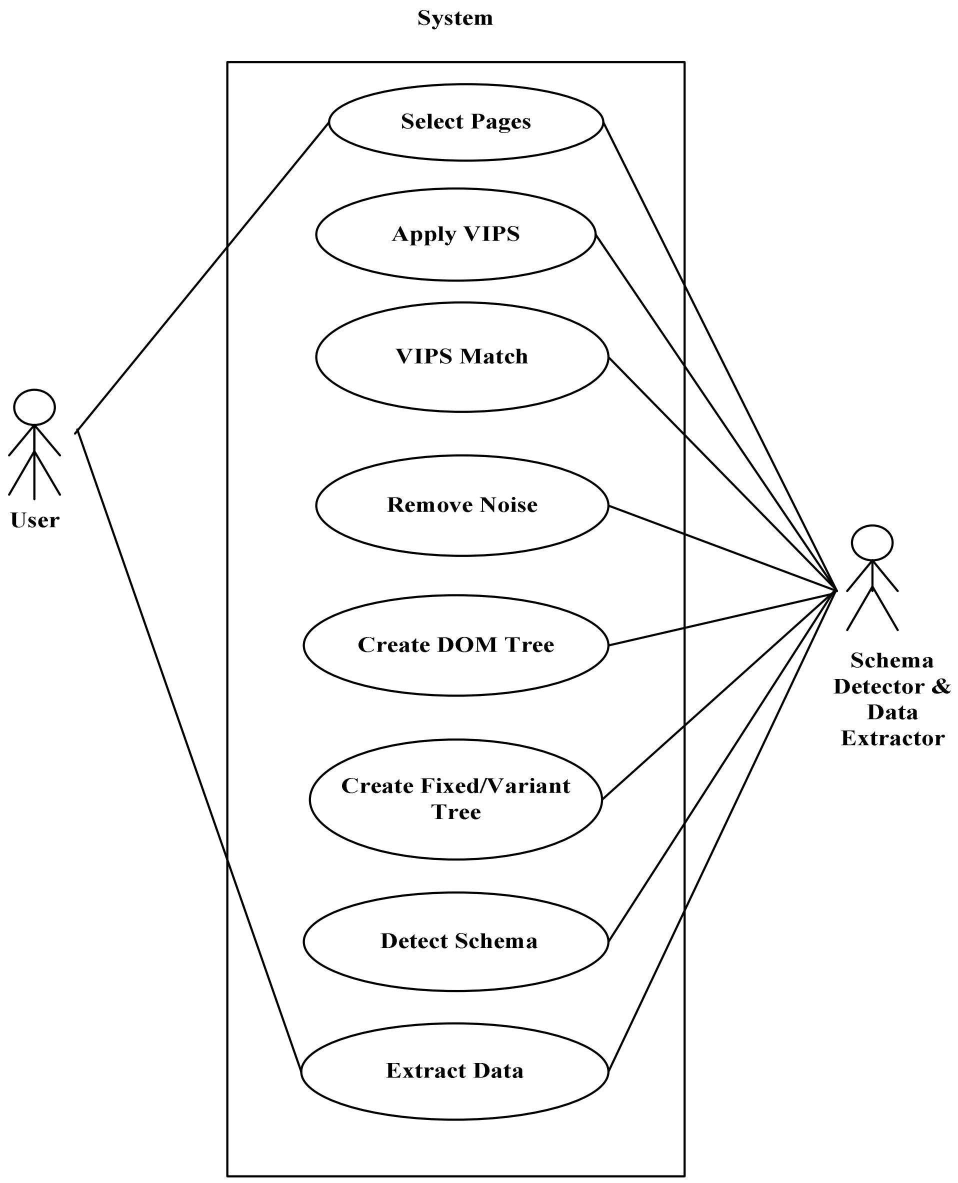


Figure 7.4 Use case diagrams for schema detection and data extraction

In fig 7.4 use cases used in this module are Select web page, Apply VIPS, VIPS match, remove noise, create DOM tree, create fix/ variant tree, detect schema and extract data This use case diagram shows interaction between user and different use cases of the module. Two actors are user and schema detector and extractor.

c) **Activity Diagram:**

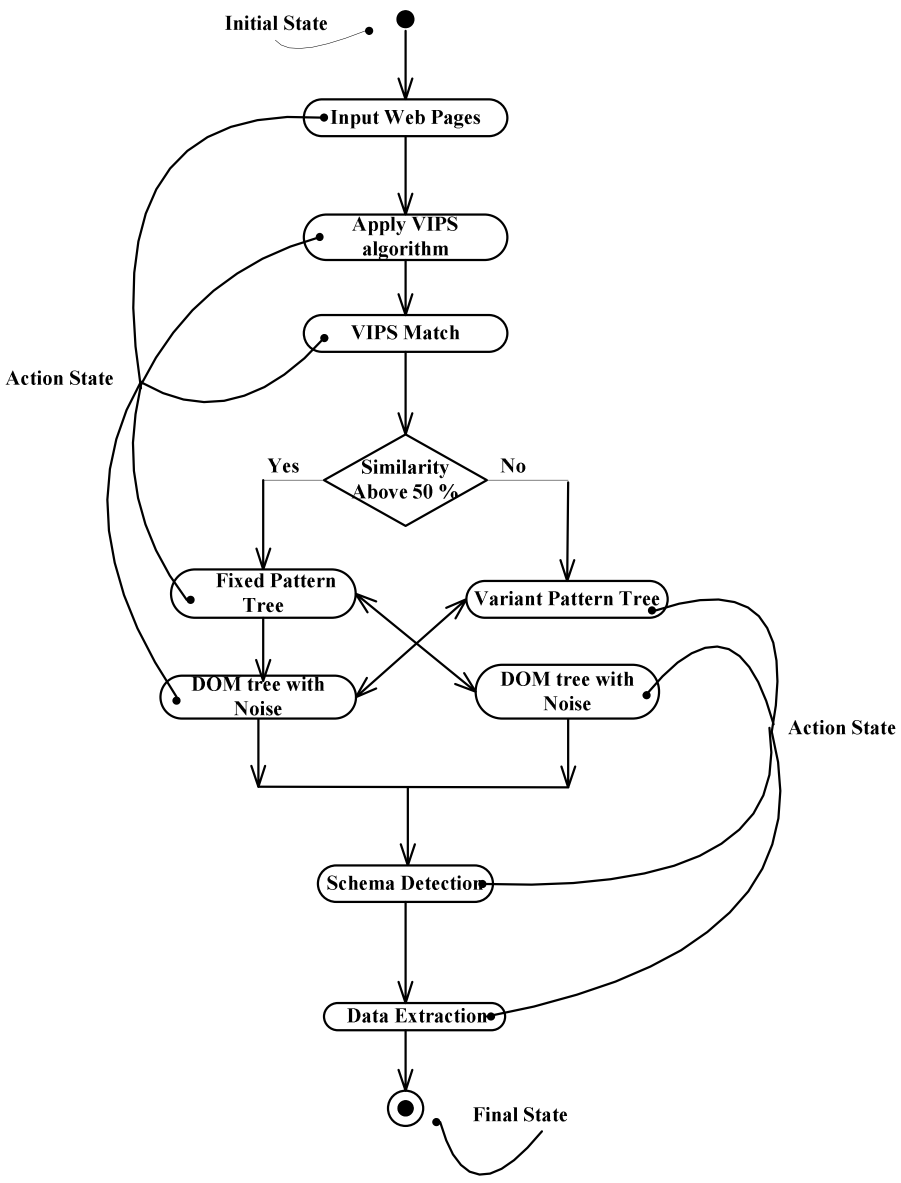


Figure 7.5 Activity diagram for schema detection and data extraction

In fig 7.5 we have used different processes viz.Input web pages , apply VIPS algrithm, VIPS match, fixed pattern tree, variant pattern tree, DOM tree with noise.and Dom tree without noise, schema detection and data extraction. Activity Diagram can be used to describe the dynamic aspects of the system.

e) State Chart Diagram:

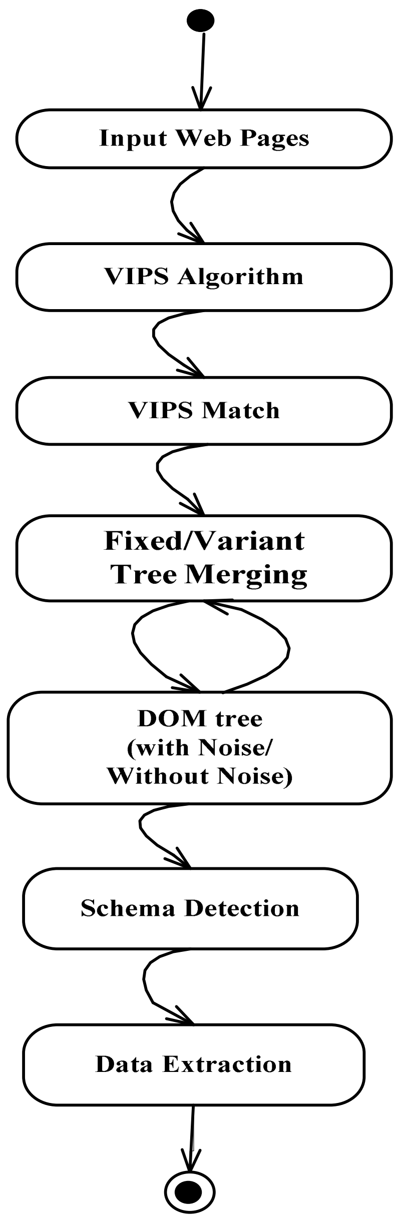


Figure 7.6 State chart diagrams for schema detection and data extraction

This fig.7.6 shows different states like Input web pages, VIPS algorithm, VIPS Match, fixed / variant tree merging, DOM tree with and without noise, schema detection and data extraction.. Sate chart diagrams are used to model the states and events operating on the system.

f) Component Diagram:

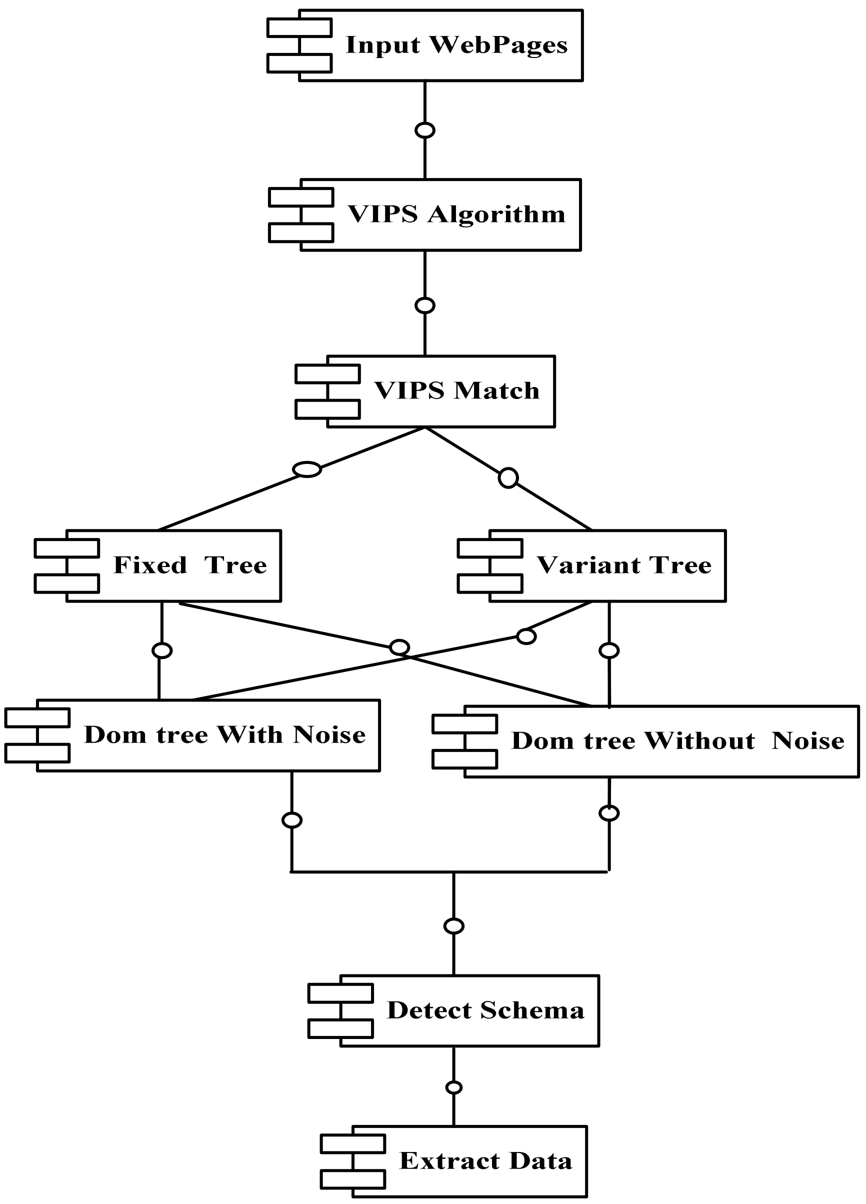


Figure 7.7 Component diagram for schema detection and data extraction

This fig. 7.7 shows different components and their communication with each other. The component diagram provides a graphical view of the dependencies and generalizations among software components. Purpose of component diagram is to describe the components used to make the functionalities. Component diagram describes Input web pages, VIPS algorithm, VIPS match, fixed tree, variant tree, DOM tree with noise and DOM tree without noise, detest schema and extract data.

g) Deployment Component Diagram:

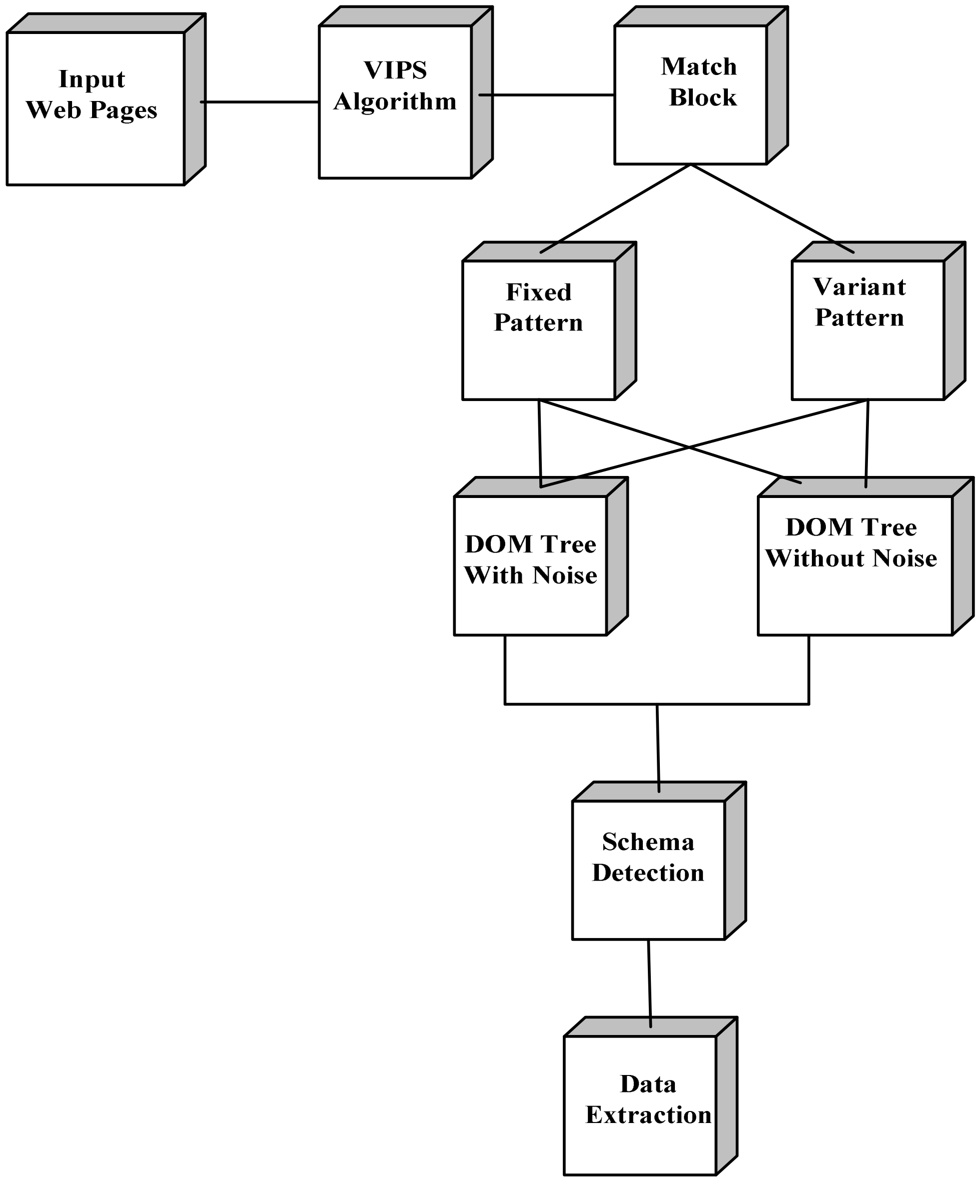


Figure 7.8 Deployment diagram for schema detection and data extraction

In fig, 7.8 some components of the system are Input Web Pages, VIPS algorithms, match block, fixed pattern, and variant pattern, DOM tree with noise, DOM tree without noise, Schema detection and data extraction. Component diagrams and deployment diagrams are closely related. Component diagrams are used to describe the components in the system and deployment diagrams shows how they are deployed in the hardware.

7.7 Snapshots:

7.7.1 Schema Detection And Data ExtractionModule:

1) Shows the screen for the module scheme detection and data extraction:

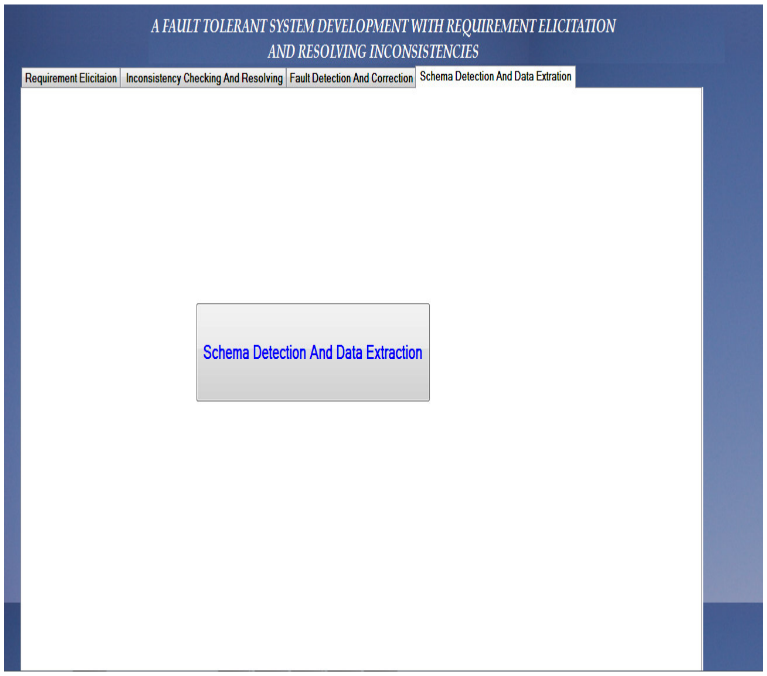

A FAULT TOLERANT SYSTEM DEVELOPMENT WITH REQUIREMENT ELICITATION AND RESOLVING INCONSISTENCIES
Requirement Elicitaion
Inconsistency Checking And Resolving
Fault Detection And Correction
Schema Detection And Data Extration
Schema Detection And Data Extraction

Figure 7.9 Shows menu of the module scheme detection and data extraction

2) Shows the index screen for the module scheme detection and data extraction:

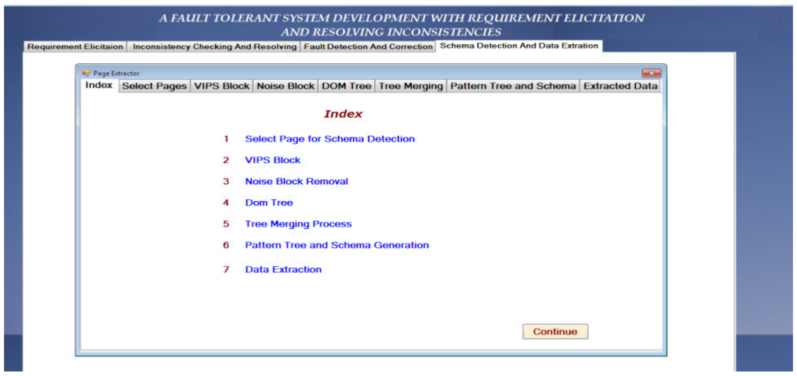

A FAULT TOLERANT SYSTEM DEVELOPMENT WITH REQUIREMENT ELICITATION AND RESOLVING INCONSISTENCIES
Requirement Elicitaion
Inconsistency Checking And Resolving
Fault Detection And Correction
Schema Detection And Data Extration
Page Extractor
Index
Select Pages
VIPS Block
Noise Block
DOM Tree
Tree Merging
Pattern Tree and Schema
Extracted Data
Index
1 Select Page for Schema Detection
2 VIPS Block
3 Noise Block Removal
4 Dom Tree
5 Tree Merging Process
6 Pattern Tree and Schema Generation
7 Data Extraction
Continue

Figure 7.10 Shows the index screen for the module scheme detection and data extraction

3) Shows the browing the input file:

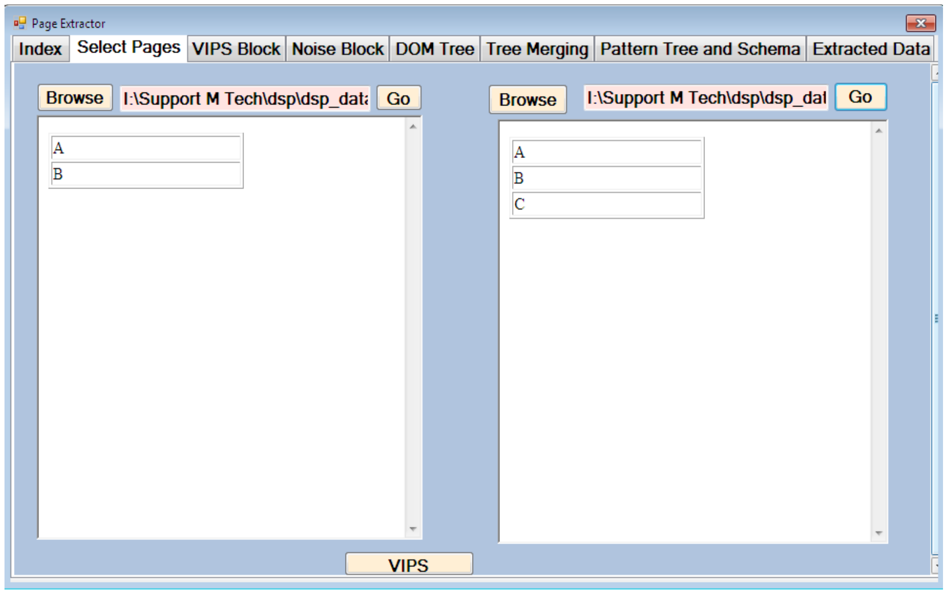


Figure 7.11 Shows, on selected web pages we apply VIPS algorithm ,and visual blocktrees of pages are constructed

4) Apply VIPS Algorithm:

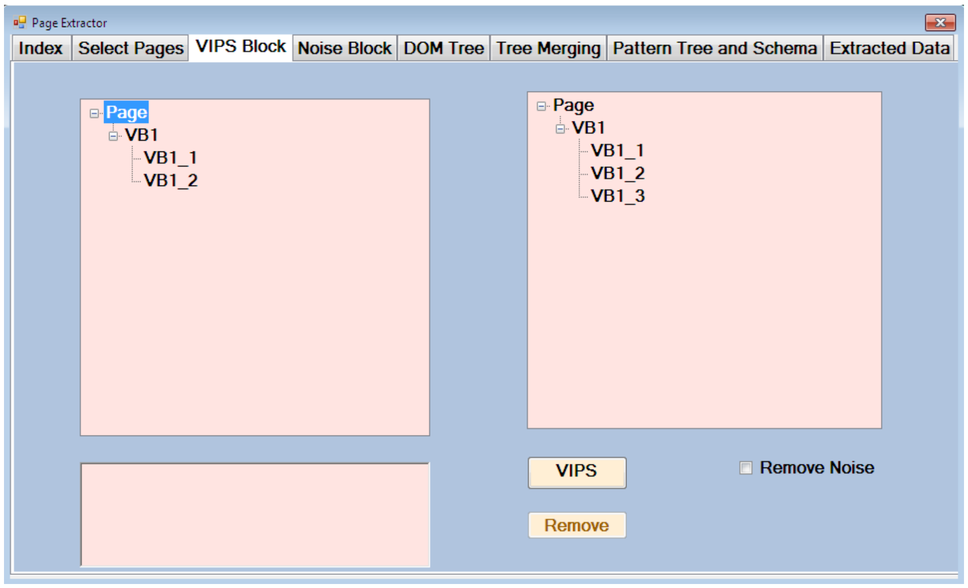


Figure 7.12 shows process of VIPS

5) Shows the creation of the DOM tree:

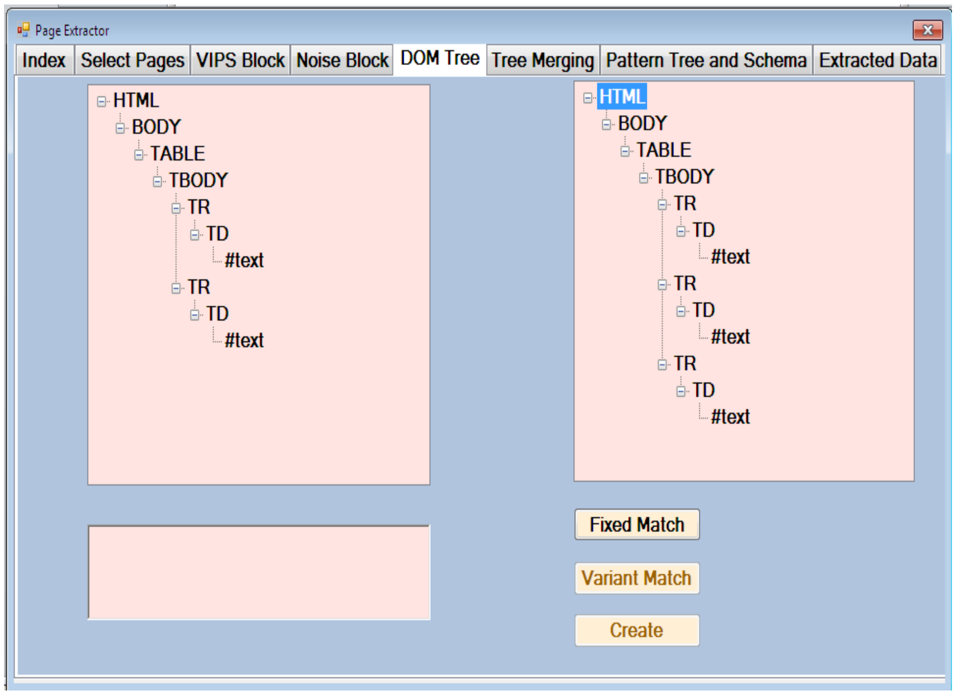
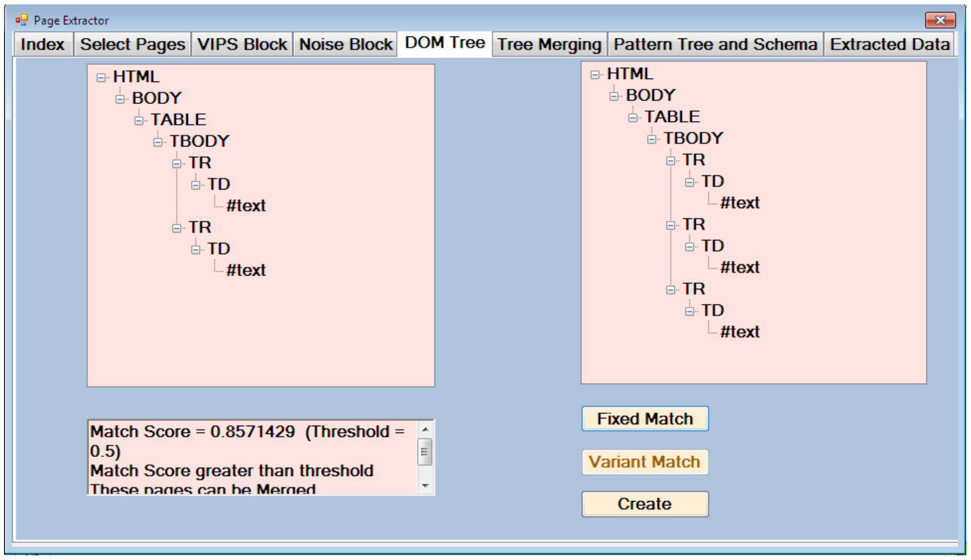


Figure 7.13 Shows the creation of the DOM tree

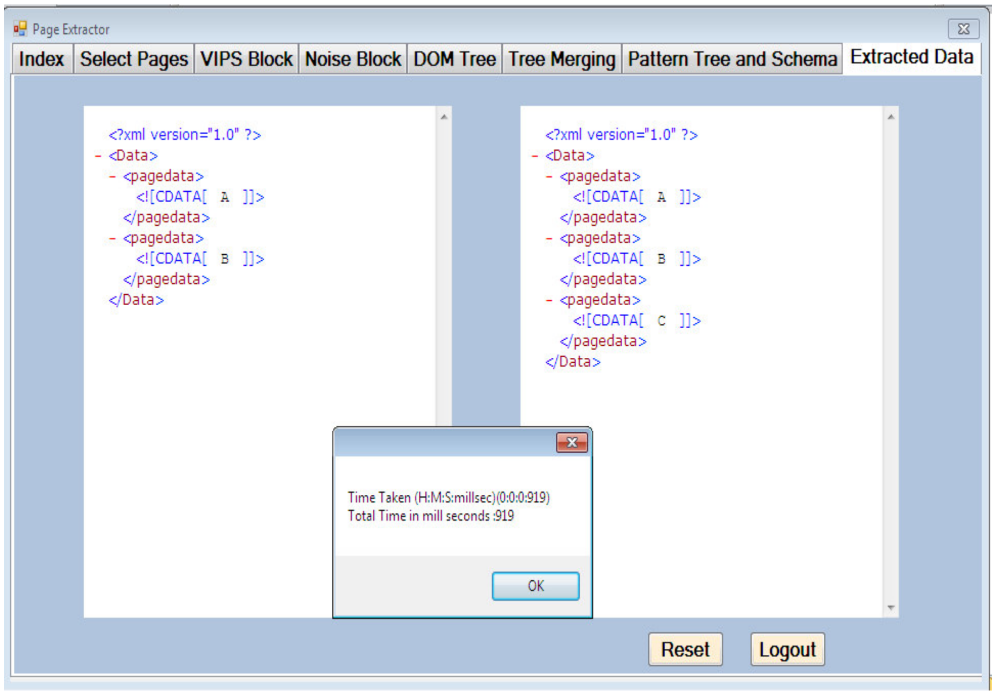
6) Shows the matching score between the trees:



```
Match Score = 0.8571429  (Threshold =
0.5)
Match Score greater than threshold
These pages can be Merged
```

Figure 7.14 Shows the matching score between the trees

7) **Shows time for the data extraction:**



```
<?xml version="1.0" ?>
- <Data>
  - <pagedata>
      <![CDATA[ A ]]>
    </pagedata>
  - <pagedata>
      <![CDATA[ B ]]>
    </pagedata>
  </Data>
```

```
<?xml version="1.0" ?>
- <Data>
  - <pagedata>
      <![CDATA[ A ]]>
    </pagedata>
  - <pagedata>
      <![CDATA[ B ]]>
    </pagedata>
  - <pagedata>
      <![CDATA[ C ]]>
    </pagedata>
  </Data>
```

```
Time Taken (H:M:S:millsec)(0:0:0:919)
Total Time in mill seconds :919
```

Figure 7.15 Shows time for the data extraction

7.8 Testing:

Table 7.1 Test cases for schema detection and data extraction

Test Cases ID	Test Case Name	Description	Step Carried	Expected Results	Actual Results	Test Case Result
TC1	Index	List of processing components	Different component having different functionality	List of Components	As expected	Pass
TC2	Browse	Browse different websites	Websites having tags and css	Website structure is displayed	As expected	Pass
TC3	VIPS Match	VIPS Match	1. Click on VIPS Match button. Start Application -> Click Login- > Index page -> Continue-	1. The result template matched or not should be displayed. 2. If user checked	As expected	Pass

Test Cases ID	Test Case Name	Description	Step Carried	Expected Results	Actual Results	Test Case Result
			>Browse-> VIPS->VIPS Match. 2 User should check checkbox Remove Noise if he wants	checkbox Remove Noise then Without Noise button should be enabled. Otherwise page with DOM trees of input pages should be displayed.		
TC4	Remove Noise Button	Click on Without Noise button	1. User should click on Without Noise button. 2. User should click on Create DOM trees button.	1. Input pages without noise blocks should be displayed. 2. DOM trees of that page should be displayed.	As expected	Pass
TC5	Fixed Match Button	Click on Fixed Match Button	User should click on Fixed Match Button.	Appropriate result should be displayed and according to that Create Pattern Tree button should be enabled.	As expected	Pass
TC6	Variant Match	Click on Variant Match	User should click on Variant Match Button	Appropriate result should be displayed and according to that Create Pattern Tree button should be displayed.	As expected	Pass
TC7	Create Pattern Tree Button	Click on Create Pattern Tree Button	User should click on Create Pattern Tree Button.	Pattern trees of pages should be displayed.	As expected	Pass
TC8	Generate Schema Button	Click on Generate Schema Button	User should click on Generate Schema Button	Schema should be generated and displayed.	As expected	Pass
TC9	Extract Data Button	Click on Extract Data Button	User should click on Extract Data Button	Extracted data should be displayed and message box representing time required to extract data should be displayed.	As expected	Pass
TC10	Logout Button	Click on Logout Button	User should click on Logout Button	Login page should be displayed.	As expected	Pass

Test Cases ID	Test Case Name	Description	Step Carried	Expected Results	Actual Results	Test Case Result
TC11	Reset Button	Click on Reset Button	User should click on Reset Button	Index page should be displayed.	As expected	Pass

7.9 Result Discussion:

For another measure of the experiment, time-based comparison is conducted between the two systems on the ten web sites. We use a system Intel Core 2 Duo i3-330 with 3GB RAM for the experiment. The fourth column shows the time consumed by the proposed system and the third column shows the time consumed by existing system in the table 7.2.

Table7.2 Time required for data extraction using Traditional system and proposed system

Id	Website	No. of Pages Tested	Traditional System	Proposed System
#001	Amazon .in	10	14196	13672
#002	www.amazon.in/Jewellery	2	82797	77172
#003	www.amazon.com/Artist	10	39028	30681
#004	*kitcoek.in/*	4	32797	30735
#005	*www.dypkopeng.ac.in*	2	2062	1938
#006	*coekolhapur.bharatividyapeeth.edu*	2	99470	38084
#007	*www.unishivaji.ac.in/*	2	10266	10047

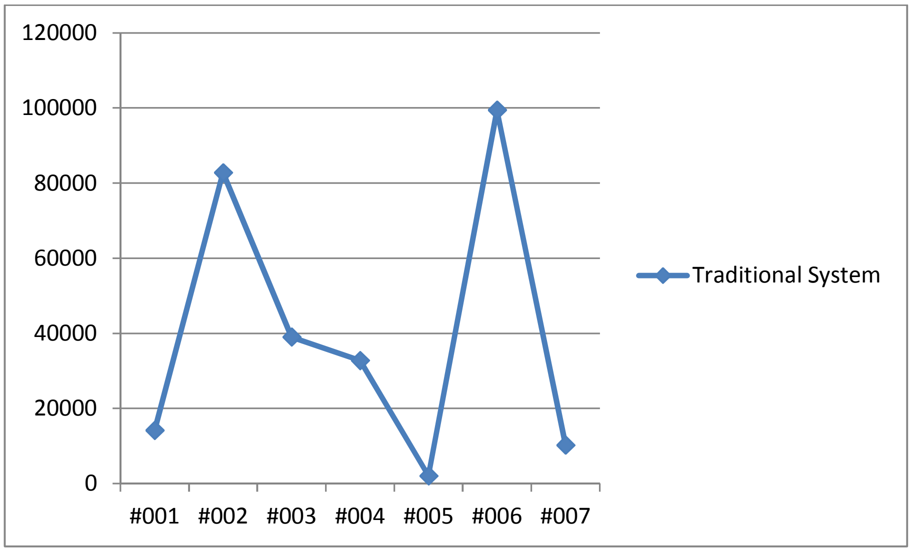


Figure 7.16 Time required for data extraction for traditional system

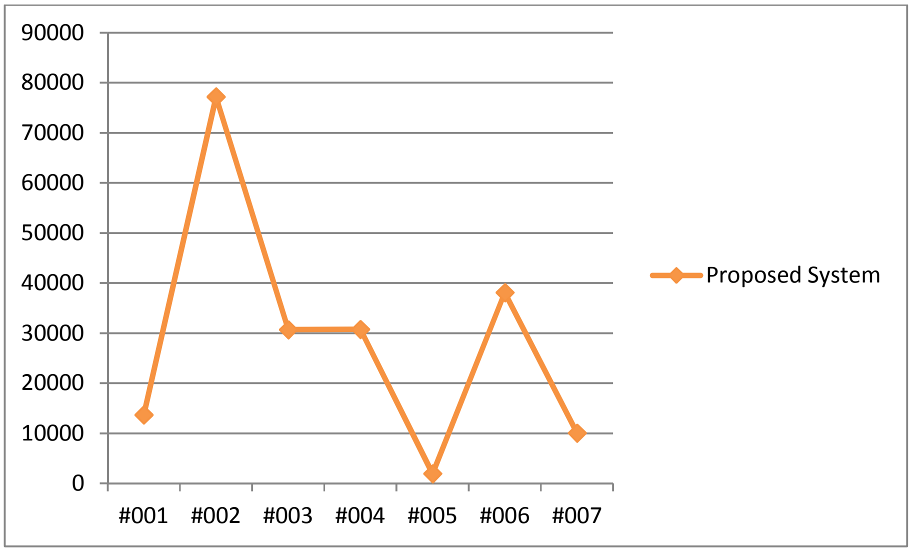


Figure 7.17 Time required for data extraction for Proposed system

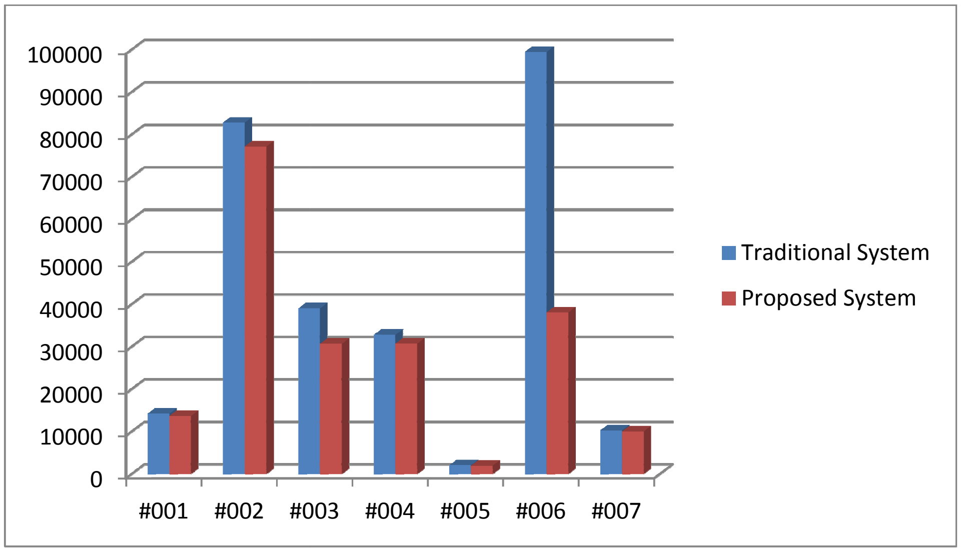


Figure 7.18 Time required for data extraction using Traditional system and proposed system

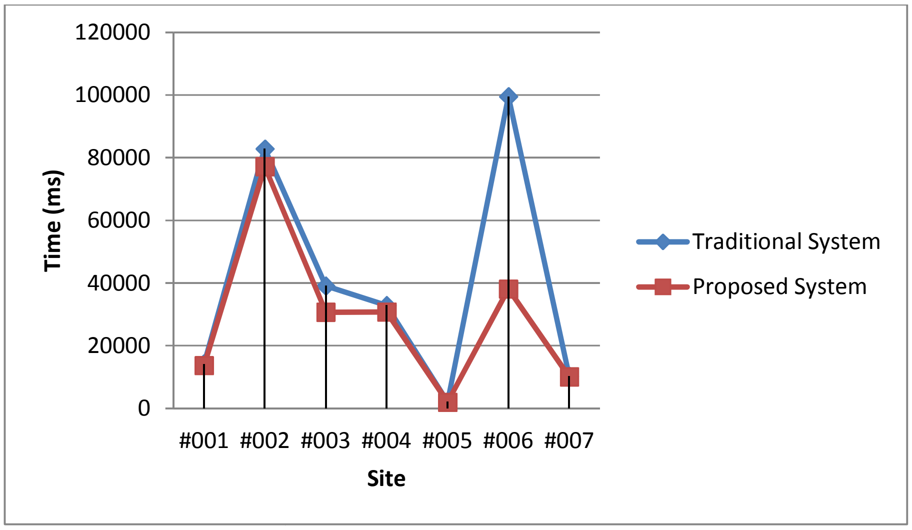


Figure 7.19 Time required for data extraction using Traditional system and proposed system

As shown in above figure, the graph represents the time analysis for the sites given in the table. Here, X-axis represents the name of website on which experiment is done and Y-axis represents time in milliseconds. Here for Y-axis value one unit is 1000 ms. Blue bar represents time required by existing system for extracting data from web pages containing noise blocks and orange bar represents time required by proposed system for extracting data from same web pages after removing noise blocks.

7.10 Pseudo code:

```
Sample Code
using System;
usingSystem.Text;
usingSystem.Collections;
usingFivaTech_Data_Extraction.common;
usingSystem.Windows.Forms;

namespaceFivaTech_Data_Extraction.fivatech
{
classPatternMining
{
privateboolprintBoolean;

privateRichTextBoxpatternText;

privateDictionary<int, int>lValueMap = newDictionary<int, int>();

privateCommonMethodscommonMethods = newCommonMethods();

publicArrayListpatternDetailsList = newArrayList();

publicArrayListorigionalAlignedList = newArrayList();

privateintpatternId = 0;

publicPatternMining(bool print, RichTextBox pattern, ArrayListorigionalList)
{
this.printBoolean = print;
this.patternText = pattern;

for (int i = 0; i <origionalList.Count; i++)
{
origionalAlignedList.Add(origionalList[i]);
}
}

publicArrayListpatternminingAlgo(ArrayListchildList, int extend)
{
int K = compLValue(childList, extend);
intst;
intnewRep;
ArrayList subList1;
ArrayList subList2;

for (int i = 1; i <= K; i++)
{
st = 0;

while ((st = Next(childList, i, st)) >= 0)
{
newRep = 0;

subList1 = newArrayList();
for (int p = st; p <= st + i - 1; p++)
{
subList1.Add(childList[p]);
}

for (int j = st + i; j + i - 1 <= childList.Count; j += i)
{

subList2 = newArrayList();

for (int q = j; q <= j + i - 1; q++)
{
if ((j + i - 1) <childList.Count)
{
subList2.Add(childList[q]);
}
else
{
break;
}
}

if (match(subList1, subList2))
{

}
else
{
break;
}
newRep++;
}

if(newRep> 0)
{
childList = modifyList(childList, st, newRep, i);
K = compLValue(childList, extend);
st += i;
}
else
{
st++;
}
}
}

if(patternCanExtend(childList, extend))
{
extend = extend + 1;
patternminingAlgo(childList, extend);
}

returnchildList;
}

publicintcompLValue(ArrayListchildList, int extend)
{
intkvalue;

booloccurenceFound;
```

```
intoccurence;

lValueMap = newDictionary<int, int>();

for (int i = 0; i <childList.Count; i++)
{
occurence = 1;

occurenceFound = false;

for (int j = i + 1; j <childList.Count; j++)
{
if (commonMethods.compareNodes(childList[i], childList[j]))
{
if (occurence == extend)
{
int l = j - i;
lValueMap.Add(i, l);
occurenceFound = true;
break;
}
else
{
occurence++;
}
}
}
if (!occurenceFound)
{
lValueMap.Add(i, 0);
}

}
kvalue = MaxLValue(lValueMap);
returnkvalue;
}

publicintMaxLValue(Dictionary<int, int>lValueMap)
{
intmaxVal = 0;

foreach (KeyValuePair<int, int>lValueinlValueMap)
{
intcurrentValue = lValue.Value;

if (currentValue>maxVal)
{
maxVal = currentValue;
}
}

returnmaxVal;
}

publicintMinLValue(Dictionary<int, int>lValueMap)
{
intminVal = 0;

foreach (KeyValuePair<int, int>lValueinlValueMap)
```

```
{
intcurrentValue = lValue.Value;

if (currentValue<minVal)
{
minVal = currentValue;
}
}
returnminVal;
}
publicint Next(ArrayListchildList, int i, intst)
{
intstartPosition = -1;

for (int f = st; f <childList.Count; f++)
{
if (lValueMap[f] == 0)
{
startPosition = -1;
}
else
{
if (lValueMap[f] == i)
{
startPosition = f;
break;
}
else
{
startPosition = -1;
}
}
}
returnstartPosition;
}

publicbool match(ArrayList subList1, ArrayList subList2)
{
boolmatchFound = false;

if (subList1.Count == subList2.Count)
{
for (int i = 0; i < subList1.Count; i++)
{
if (commonMethods.compareNodes(subList1[i], subList2[i]))
{
matchFound = true;
}
else
{
matchFound = false;
break;
}
}
}
returnmatchFound;
}
publicArrayListmodifyList(ArrayListchildList, intst, intnewRep, int i)
{
ArrayListpatternList = newArrayList();
```

```
if (printBoolean)
{
StringorigionalList = "";

for (int g = 0; g <childList.Count; g++)
{
origionalList = origionalList + "," + commonMethods.getNodeSymbol(childList[g]);
}
patternText.Text = patternText.Text + "Origional ->" + origionalList + "\n";
}
string pattern = "";
patternId = patternId + 1;
intdeleteIndex;
for (int x = 0; x < (newRep * i); x++)
{
if (x < i)
{
pattern = pattern + "," + commonMethods.getNodeSymbol(childList[st + x]);
patternList.Add(childList[st + x]);
commonMethods.updateNodeInfoForPattern(childList[st + x], patternId);
}

deleteIndex = st + i;
childList.RemoveAt(deleteIndex);
}

if (printBoolean)
{
StringmodifiedList = "";
patternText.Text = patternText.Text + "Patten ->" + pattern + "\n";

for (int g = 0; g <childList.Count; g++)
{
modifiedList = modifiedList + "," + commonMethods.getNodeSymbol(childList[g]);
}

patternText.Text = patternText.Text + "Modified ->" + modifiedList + "\n";
patternText.Text = patternText.Text + "---------------------------------- \n";
}
PatternDetailspatternDetail = newPatternDetails(patternId, st, newRep, patternList, i);
patternDetailsList.Add(patternDetail);
// Console.WriteLine("pattern ---" + pattern);
returnchildList;
}
publicboolpatternCanExtend(ArrayListchildList, int extend)
{
boolpatternExtend = false;
intshortestPattern = 2 * MaxLValue(lValueMap) * (extend + 1);
if (childList.Count<shortestPattern)
{
patternExtend = true;
}
returnpatternExtend;
}

}
}
//schema2
using System;
```

```
usingSystem.Xml;
using System.IO;

namespaceFivaTech_Data_Extraction.fivatech
{
classExtractData
{
publicExtractData()
{

}
privatestaticvoidTraverseTree(TreeNodetn)
{
for (int i = 0; i <tn.Nodes.Count; i++)
{
string name = tn.Nodes[i].Text;

TraverseTree(tn.Nodes[i]);
}

string text = tn.Text;
}
privatestaticstring data;


publicstaticvoidTraverseChild(TreeNodePageDomTree, TreeNodeSchemasTree)
{
//Traverse Dom Tree

if (PageDomTree.Nodes.Count> 0)
{
for (int i = 0; i <PageDomTree.Nodes.Count; i++)
{
FVNodeInfotagData = (FVNodeInfo)PageDomTree.Tag;
TraverseChild(PageDomTree.Nodes[i], SchemasTree);
}
}
else
{
FVNodeInfonodeInfo = (FVNodeInfo)PageDomTree.Tag;
if (nodeInfo.basic_subtype == "VARIANT"andand !nodeInfo.nodeText.Trim().Equals(string.Empty))
{
streamWriter.WriteLine("<pagedata><![CDATA[ " + nodeInfo.nodeText + "]]></pagedata>"); //
tagData.nodeText
}
// data += "<td>"+nodeInfo.nodeText+"</td>";
}
/*
if (data != "") data += "\r\n";
string element = SchemasTree.Text;
if (element.Contains("<![CDATA[ "))
{
element = element.Replace("<![CDATA[", "");
element = element.Replace("]]>", "");
}
else
{
FVNodeInfotagData = (FVNodeInfo)PageDomTree.Tag;
data += tagData.nodeText;
if (PageDomTree.Nodes.Count> 0)
{
for (int i = 0; i <PageDomTree.Nodes.Count; i++)
{
try
{
TraverseChild(PageDomTree.Nodes[i], SchemasTree.Nodes[i]);
}
catch (Exception expt) { }
}
}
else
{
//find this node in PageDomTree
FVNodeInfo tagData1 = (FVNodeInfo)PageDomTree.Tag;

data += "<![CDATA[ " + tagData1.nodeText + "]]>";

}
}
/*
for (int i = 0; i <SchemasTree.Nodes.Count; i++)
{
string type = SchemasTree.Nodes[i].Text;
string element = PageDomTree.Nodes[i].Text;
//if (PageDomTree.Nodes[i].Nodes.Count> 1)
{
TraverseChild(PageDomTree.Nodes[i], SchemasTree.Nodes[i]);
}
}*/

//TraverseTree(PageDomTree);
//TraverseTree(SchemasTree);
}
privatestaticStreamWriterstreamWriter;

publicstaticvoidExtractDataFromTemplate(stringfilename,TreeNodePageDOMTree, TreeNodeSchemasTree)
{
streamWriter = newStreamWriter(filename, false, System.Text.Encoding.UTF8);
//Write the header
streamWriter.WriteLine("<?xml version=\"1.0\"?>");
//data = "<html>";
//data += "<table>";
//Extract Data From DOM using the Schemas
streamWriter.WriteLine("<Data>");
TraverseChild(PageDOMTree, SchemasTree);
streamWriter.WriteLine("</Data>");
// data += "</table>";
///data += "</html>";

streamWriter.Close();

}
}
}

//schema3
using System;
usingSystem.Text;
usingSystem.Windows.Forms;
usingmshtml;
```

```
namespaceFivaTech_Data_Extraction.fivatech
{
classFilterTemplateBlock
{
privateTreeNode template;
privatefloatnodeAreaBody = 0.0f;
privatecommon.VIPSCommonvipsCommon;
publicFilterTemplateBlock()
{
vipsCommon = newFivaTech_Data_Extraction.common.VIPSCommon();
}

privatefloat Area(IHTMLDOMNode element1)
{
try
{
IHTMLElement element = (IHTMLElement)element1;

long width=0;
long height=0;

//vipsCommon.getWebPageSize(element, out width,out height);
//HTMLDocumentClasscls = element.document;
width = element.offsetHeight;
height = element.offsetWidth;

return (float)width * height;
}
catch (Exceptionexpt) { }

return 0;
}
privatefloatPercentageArea(floatnodeArea)
{
floatpercentageArea = 0.0f;
percentageArea = 100.0f * (nodeArea / nodeAreaBody);
/*while (percentageArea< 2)
{
percentageArea *= 10;
}*/
returnpercentageArea;
}

privatelongpreWidth = 0;
privatelongpreHeight = 0;
privatevoidRemoveImages(IHTMLDOMNode parent)
{
if (parent.hasChildNodes())
{
IHTMLDOMChildrenCollectionallchild = (IHTMLDOMChildrenCollection)parent.childNodes;

int length = allchild.length;

for (int i = 0; i < length; i++)
{
IHTMLDOMNodechild_node = (IHTMLDOMNode)allchild.item(i);

try
{
```

```
IHTMLElement element = (IHTMLElement)child_node;
long width = 0;
long height = 0;
//area = element.offsetHeight * element.offsetWidth;
vipsCommon.getWebPageSize(element, out width, out height);

if (element.tagName.ToLower().Equals("img") || element.tagName.ToLower().Equals("a"))
{
//if (preWidth == 0 andandpreWidth == 0)
{
preHeight = height;
preWidth = width;
}
if (preHeight == height andandpreWidth == width )
{
float area = Area(child_node);
floatpercentageArea = PercentageArea(area);
if (element.tagName.ToLower().Equals("a") andandpercentageArea> 0.4)
continue;
//if (percentageArea< 40)
{
child_node.removeNode(true);
i--;
length--;
}
}
else
{
preWidth = width;
preHeight = height;
}
}
else
{
RemoveImages(child_node);
}
}
catch (Exceptionexpt) { }
}
}
}
privatevoid Traverse(IHTMLDOMNodeparentnode)
{
if (parentnode.hasChildNodes())
{
IHTMLDOMChildrenCollectionallchild = (IHTMLDOMChildrenCollection)parentnode.childNodes;

int length = allchild.length;

intIndexBiggestChildNode = 0;
floatMaxPercentageArea = 0.0f;

for (int i = 0; i < length; i++)
{
IHTMLDOMNodechild_node = (IHTMLDOMNode)allchild.item(i);
float area = Area(child_node);
if (area < 0.40) continue;
floatpercentageArea = PercentageArea(area);

if (percentageArea>MaxPercentageArea)
```

```
{
MaxPercentageArea = percentageArea;
IndexBiggestChildNode = i;
}
}

for (int i = 0; i < length; i++)
{
IHTMLDOMNodechild_node = (IHTMLDOMNode)allchild.item(i);
if (i != IndexBiggestChildNode)
{
//Remove Matching Image nodes
RemoveImages(child_node);
}
else
{
Traverse(child_node);
}
}


}
return;
}


publicvoidFilteringTemplateBlock(AxSHDocVw.AxWebBrowseraxWebBrowser,string filename)
{

IHTMLDocument3 HTMLDocument = (IHTMLDocument3)axWebBrowser.Document;
IHTMLDOMNoderootDomNode = (IHTMLDOMNode)HTMLDocument.documentElement;

nodeAreaBody = Area(rootDomNode);

Traverse(rootDomNode);
// axWebBrowser.Refresh();//  = axWebBrowser.Document;

common.CommonMethods common = newFivaTech_Data_Extraction.common.CommonMethods();
common.ExportToHtml(rootDomNode, filename);
}
}
}


//schema 4
using System;
usingSystem.Text;
usingSystem.Windows.Forms;
usingFivaTech_Data_Extraction.fivatech;

namespaceFivaTech_Data_Extraction.fivatech
{
classGenerateSchema
{
publicintmarkTheOrder(TreeNoderootNode)
{
FVNodeInfonodeInfo = (FVNodeInfo)rootNode.Tag;

if (rootNode.Nodes.Count == 0)
{
```

```
if ("FIXED".CompareTo(nodeInfo.basic_subtype) == 0)
{
return 0;
}
else
{
return 1;
}
}

if (rootNode.Nodes.Count == 1)
{
if ("SET".CompareTo(nodeInfo.typeConstructor) == 0 || "OPTIONAL".CompareTo(nodeInfo.typeConstructor)
== 0)
{
nodeInfo.typeOrder = "order-1-" + nodeInfo.typeConstructor;
}
else
{
markTheOrder(rootNode.Nodes[0]);
}

}
int k = 0;
for (int i = 0; i <rootNode.Nodes.Count; i++)
{
k += markTheOrder(rootNode.Nodes[i]);
}

if (k == 0)
{
return 0;
}
if ("VIRTUAL".CompareTo(nodeInfo.nodeType) == 0)
{
nodeInfo.typeOrder = "order-" + k + "-" + nodeInfo.typeConstructor;
}
else
{
```

```
if (k == 1)
{
if ("SET".CompareTo(nodeInfo.typeConstructor) == 0 || "OPTIONAL".CompareTo(nodeInfo.typeConstructor)
== 0)
{
nodeInfo.typeOrder = "tuple-" + k + "-"+ nodeInfo.typeConstructor;
}
}
else
{

nodeInfo.typeOrder = "tuple-" + k + "-" + nodeInfo.typeConstructor;
}


}
return 1;
}

// To be Removed Test Data //////////////////////////////////////////////////////

publicTreeNode getPatternTree1()
{
TreeNoderootNode, node1, node2,  node4, node5;

FVNodeInfofvNodeInfo = newFVNodeInfo("HTML", "ACTUAL", "BASIC", "FIXED", "HTML");
rootNode = newTreeNode(fvNodeInfo.nodeSymbol);
rootNode.Tag = fvNodeInfo;

fvNodeInfo = newFVNodeInfo("BODY", "ACTUAL", "BASIC", "FIXED", "BODY");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
rootNode.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("TABLE", "ACTUAL", "BASIC", "FIXED", "TABLE");
node2 = newTreeNode(fvNodeInfo.nodeSymbol);
node2.Tag = fvNodeInfo;
node1.Nodes.Add(node2);

fvNodeInfo = newFVNodeInfo("{TR}", "VIRTUAL", "SET", "", "TR");
```

```
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node2.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("TD", "ACTUAL", "BASIC", "FIXED", "TD");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node1.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("(TD)", "VIRTUAL", "OPTIONAL", "", "TD");
node5 = newTreeNode(fvNodeInfo.nodeSymbol);
node5.Tag = fvNodeInfo;
node1.Nodes.Add(node5);

fvNodeInfo = newFVNodeInfo("B", "ACTUAL", "BASIC", "FIXED", "B");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node4.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("*", "ACTUAL", "BASIC", "VARIANT", "TEXT");
node2 = newTreeNode(fvNodeInfo.nodeSymbol);
node2.Tag = fvNodeInfo;
node1.Nodes.Add(node2);

fvNodeInfo = newFVNodeInfo("*", "ACTUAL", "BASIC", "VARIANT", "TEXT");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node4.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("BR", "ACTUAL", "BASIC", "FIXED", "BR");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node4.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("(V)", "VIRTUAL", "OPTIONAL", "", "V");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node4.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("AVG", "ACTUAL", "BASIC", "FIXED", "TEXT");
```

```
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node1.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("B", "ACTUAL", "BASIC", "FIXED", "B");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node1.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("*", "ACTUAL", "BASIC", "VARIANT", "TEXT");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node4.Nodes.Add(node1);
// -----------------------------------------------------

fvNodeInfo = newFVNodeInfo("B", "ACTUAL", "BASIC", "FIXED", "B");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node5.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("LST", "ACTUAL", "BASIC", "FIXED", "TEXT");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node4.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("SPAN", "ACTUAL", "BASIC", "FIXED", "SPAN");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node5.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("*", "ACTUAL", "BASIC", "VARIANT", "TEXT");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node4.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("B", "ACTUAL", "BASIC", "FIXED", "B");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node5.Nodes.Add(node4);
```

```
fvNodeInfo = newFVNodeInfo("SAVE", "ACTUAL", "BASIC", "FIXED", "TEXT");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node4.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("FONT", "ACTUAL", "BASIC", "FIXED", "FONT");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node5.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("*", "ACTUAL", "BASIC", "VARIANT", "TEXT");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node4.Nodes.Add(node1);

returnrootNode;
}

publicTreeNodegetPatternTree()
{
TreeNoderootNode, node1, node2, node3, node4, node5, node6;

FVNodeInfofvNodeInfo = newFVNodeInfo("HTML", "ACTUAL", "BASIC", "FIXED", "HTML");
rootNode = newTreeNode(fvNodeInfo.nodeSymbol);
rootNode.Tag = fvNodeInfo;

fvNodeInfo = newFVNodeInfo("BODY", "ACTUAL", "BASIC", "FIXED", "BODY");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
rootNode.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("TABLE", "ACTUAL", "BASIC", "FIXED", "TABLE");
node2 = newTreeNode(fvNodeInfo.nodeSymbol);
node2.Tag = fvNodeInfo;
node1.Nodes.Add(node2);

fvNodeInfo = newFVNodeInfo("{V1}", "VIRTUAL", "SET", "", "V1");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node2.Nodes.Add(node1);
```

```
fvNodeInfo = newFVNodeInfo("TR1", "ACTUAL", "BASIC", "FIXED", "TR1");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node1.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("TD1", "ACTUAL", "BASIC", "FIXED", "TD1");
node5 = newTreeNode(fvNodeInfo.nodeSymbol);
node5.Tag = fvNodeInfo;
node4.Nodes.Add(node5);

fvNodeInfo = newFVNodeInfo("TABLE1", "ACTUAL", "BASIC", "FIXED", "TABLE1");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node5.Nodes.Add(node4);

node5 = node4;
fvNodeInfo = newFVNodeInfo("TR", "ACTUAL", "BASIC", "FIXED", "TR");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node5.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("TD2", "ACTUAL", "BASIC", "FIXED", "TD2");
node5 = newTreeNode(fvNodeInfo.nodeSymbol);
node5.Tag = fvNodeInfo;
node4.Nodes.Add(node5);

fvNodeInfo = newFVNodeInfo("BR1", "ACTUAL", "BASIC", "FIXED", "BR1");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node5.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("A", "ACTUAL", "BASIC", "FIXED", "A");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node5.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("*", "ACTUAL", "BASIC", "VARIANT", "TEXT");
node6 = newTreeNode(fvNodeInfo.nodeSymbol);
node6.Tag = fvNodeInfo;
```

```
node4.Nodes.Add(node6);

fvNodeInfo = newFVNodeInfo("BR2", "ACTUAL", "BASIC", "FIXED", "BR2");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node5.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("SPAN", "ACTUAL", "BASIC", "FIXED", "SPAN");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node5.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("*", "ACTUAL", "BASIC", "VARIANT", "TEXT");
node6 = newTreeNode(fvNodeInfo.nodeSymbol);
node6.Tag = fvNodeInfo;
node4.Nodes.Add(node6);

fvNodeInfo = newFVNodeInfo("(V2)", "VIRTUAL", "OPTIONAL", "", "V2");
node6 = newTreeNode(fvNodeInfo.nodeSymbol);
node6.Tag = fvNodeInfo;
node4.Nodes.Add(node6);

fvNodeInfo = newFVNodeInfo("BR3", "ACTUAL", "BASIC", "FIXED", "BR3");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node6.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("*", "ACTUAL", "BASIC", "VARIANT", "TEXT");
node4 = newTreeNode(fvNodeInfo.nodeSymbol);
node4.Tag = fvNodeInfo;
node6.Nodes.Add(node4);

fvNodeInfo = newFVNodeInfo("TR2", "ACTUAL", "BASIC", "FIXED", "TR2");
node3 = newTreeNode(fvNodeInfo.nodeSymbol);
node3.Tag = fvNodeInfo;
node1.Nodes.Add(node3);

fvNodeInfo = newFVNodeInfo("TD3", "ACTUAL", "BASIC", "FIXED", "TD3");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
```

```
node3.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("{V3}", "VIRTUAL", "SET", "", "V3");
node3 = newTreeNode(fvNodeInfo.nodeSymbol);
node3.Tag = fvNodeInfo;
node1.Nodes.Add(node3);

fvNodeInfo = newFVNodeInfo("*", "ACTUAL", "BASIC", "VARIANT", "TEXT");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node3.Nodes.Add(node1);

fvNodeInfo = newFVNodeInfo("BR4", "ACTUAL", "BASIC", "FIXED", "BR4");
node1 = newTreeNode(fvNodeInfo.nodeSymbol);
node1.Tag = fvNodeInfo;
node3.Nodes.Add(node1);

returnrootNode;
}

}
}
```

7.11 Conclusion:

The goal of research work is to provide novel Web data extraction method to difficulty of page-level data extraction. This method uses visual information & DOM tree to extract template automatically from web pages .We use VIPS algorithm which is used to decide fixed/variant template pages. Proposed Approach consists of three phases. In phase 1, VIPS algorithm is applied on each page to determine fixed/variant template & data region is filtered out by removing noise blocks. In phase 2, tree merging is done to construct fixed/variant pattern tree and in phase 3, Schema & template is detected based on the pattern tree. Here multiple string alignment algorithm is used which takes optional- and set-type data into consideration. Pattern tree is constructed with which we can easily deduce template & schema for input web pages. Although many unsupervised approaches have been proposed for Web data extraction, very few works (RoadRunner and EXALG) solve this problem at a page level. Merged pattern tree gives very good result for schema and template detection.

The data extraction from web pages is done effectively using proposed approach. It is concluded that proposed system is efficient than existing system as it requires optimum time for data extraction.