Transforming Web Pages to Become Standard-Compliant through Reverse Engineering

Benfeng Chen
Computer Science Department
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
bfchen@cs.ust.hk

Vincent Y. Shen
Computer Science Department
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
shen@cs.ust.hk

ABSTRACT
Developing Web pages following established standards can make the information more accessible, their rendering more efficient, and their processing by computer applications easier. Unfortunately, more than 95% of the existing Web pages today are not "valid" in that they do not follow some of the recommendations (standards) of the World Wide Web Consortium (W3C). Fixing any Web page to make it standard-compliant is a major undertaking. There is now an open-source tool called HTML Tidy which will attempt to fix the invalid HTML code automatically. However, Tidy often changes the Web page's appearance after processing. It is not an effective tool to transform existing Web pages to make them standard-compliant.

In this paper we report the design and implementation of PURE, a tool that cleans up an HTML document through reverse engineering. PURE starts with the rendering result of a given Web page and generates valid HTML code and CSS automatically to produce the same appearance. It is found to be effective for many existing Web pages. A prototype is now available for public testing and comments.

Categories and Subject Descriptors
H.4.3 [**Communications Applications**]: Information Browsers

General Terms
Algorithms, Measurement, Performance, Design, Experimentation, Standardization.

Keywords
W3C recommendations, Web page, HTML, HTML Tidy, Cascade Style Sheets, rendering engine, browser.

1. INTRODUCTION
The World Wide Web Consortium (W3C) is an organization that develops and promotes the use of standards on the Web. However, as much as 95% of existing Web pages today are "invalid" [9] in that they fail to conform to the W3C "recommendations" (commonly considered "Web standards" by the community) published since 1995 [16]. These invalid Web pages may not be rendered consistently across platforms or by different browsers, may not be accessible to some users, and may cause problems for Web-based applications. The seriousness of the problem spawned the Web Standards Project (WaSP) [10], which was established to encourage people to design Web pages which conform to standards in order to reduce the cost and complexity of development, while increasing the accessibility and long-term viability of any site published on the Web [21].

W4A at WWW2006, 23rd-26th May 2006, Edinburgh, UK

The basic idea of Web standards is to develop Web pages with valid HTML or XHTML code and to separate content from presentation. A Web page is composed of three parts: content, presentation and behavior. According to Web standards, the content should be written in valid HTML or XHTML code; the presentation (e.g., layout, font, color, etc.) should be specified by valid CSS code; and the behavior should be controlled by valid JavaScript (officially, the ECMAScript [2]) through the DOM interface [17]. Figure 1 shows the "trinity" of Web standards [21]. It is necessary for a Web page to be compliant with Web standards because of the following reasons [5][11]:

- Its accessibility is wider because the CSS-based presentation is more flexible for different devices, browsers or handicapped people.
- Its size is smaller and bandwidth for access is reduced. For example, ESPN.com is saving 2TB of traffic per day by redesigning Web pages according to Web standards [7].
- It is friendly for machine processing. Without the misused table and font tags for layout and appearance in the code, other computers can "understand" the Web page's content better.

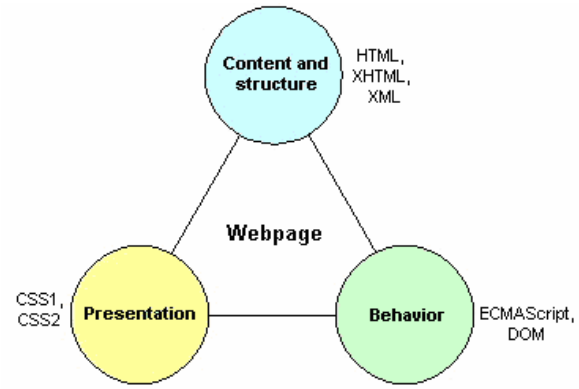

Figure 1. The trinity of Web standards

However, as we noted earlier, most Web pages on the Internet today are not standards-compliant [9]. A very common problem is that the HTML TABLE element is widely used for page layout. Note that the TABLE element is intended to mark up truly tabular information ("data tables"). But, due to its flexibility, content developers often use the TABLE element to layout pages ("layout tables"). They should have used the Cascade Style Sheets (CSS) to layout pages for all the presentation effects [12]. People believe that about 99% of existing websites are obsolete as far as Web standards are concerned [21]. Rewriting these legacy Web pages will incur a huge cost of time and money. Therefore, an automated tool will be very useful if it can help Web developers transform existing Web pages to those that are standards-compliant while keeping the appearance consistent with the original.

Fixing an existing HTML document to make it standard-compliant automatically is a difficult problem. The W3C has endorsed an open-source tool called "HTML Tidy" which can automatically fix a Web page's invalid code [8]. But it does not convert presentation-related elements from existing HTML code to using style sheets. Furthermore, it often fails when fixing a complicated Web page, especially if its layout is designed with nested HTML tables. Unfortunately, nested layout tables are widely used in legacy Web pages and various tricks are frequently used to extend the presentation effect of tables. A quick study shows that of the 500 most popular websites [1], about 470 of them use the TABLE element to control the layout. The appearance of these Web pages often changes drastically after they are processed by Tidy. Figure 2 shows an example (http://www.imdb.com/, No. 46 of the top 500 most popular websites on September 1, 2005): Figure 2 (a) is the original page and Figure 2 (b) shows parts of the resulting page after processing by Tidy, which produced a long thin page. This is indeed an extreme example but, since many of the Web pages of the top 500websites are quite complicated, Tidy messes up the appearance of at least 80% of them.

Figure 2 (a). Original Web page

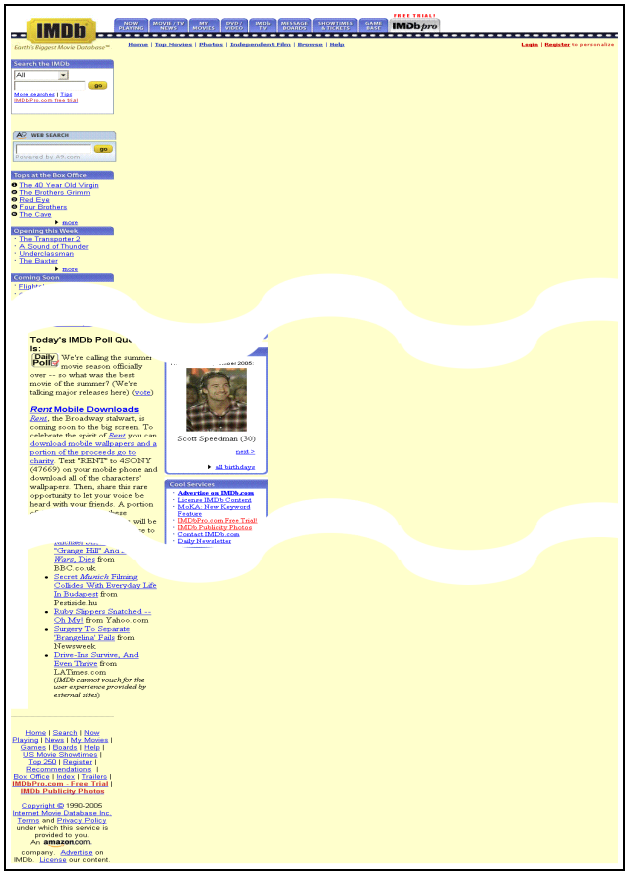

Figure 2 (b). Parts of the result after processing by HTML Tidy

This paper presents an automated tool called "PURE" ("Page clean-Up through Reverse Engineering") which uses the reverse engineering approach to transform legacy Web pages to make them compliant with Web standards while keeping the appearance consistent with the original. PURE first uses a popular Web browser to render the original Web page to be processed (the "preprocessing" step). It then examines the layout of the Web page presented in the browser's window. It segments the original Web page's content into small rectangles (called "boxes" thereafter). It retrieves the position and size of each box through the browser-generated page structure (in the form of a DOM tree [17]). A recursive algorithm is designed to reconstruct the layout of the Web page using the CSS box model [15]. At the end of this step a new Web page is generated which contains a number of empty boxes (the "layout reconstruction" step). In the last step, PURE fills in the content for each box. It takes the source code for each box from the original Web page, transforms it into standard-compliant code, and then puts the resulting code into the corresponding box in the new page (the "box filling" step). The home page of www.imdb.com after processing by PURE is shown in Figure 3.

Figure 3. Result after processing by PURE

Our preliminary evaluation shows that over 50% of the home pages of the top 500 websites can be successfully transformed through this three-step approach automatically. The failures are mostly due to some special features of the browser-generated page structure which is inconsistent with the DOM tree as defined by W3C. Most of these problems could be fixed with some human interaction. We therefore believe that PURE can become a useful tool to help Web developers to transform legacy Web pages.

The highlights of our approach are:

- We are able to separate the presentation from content. To understand a Web page's presentation from its source code is a very difficult task unless we build a rendering engine ourselves. In our reverse engineering approach, we can obtain the presentation from a popular browser (such as the Microsoft Internet Explorer (IE)). The major task is to build the same presentation using CSS, which is much easier than building a rendering engine.

- We avoid the difficulty of parsing the original HTML code by getting the HTML DOM tree from the chosen browser.

- We avoid the difficulty of handling ambiguous code and guessing the author's intention. The invalid code is often ambiguous. For example, the developer may have left out certain closing tags and the browsers must guess where to place them if some are missing. It is reasonable to assume that the authors have tested the Web pages using IE before they upload them to a Web server. So we could use IE's rendering result as the starting point for reverse engineering.

- We use a divide-and-conquer strategy by first segmenting the Web page into small boxes and then reconstructing these boxes separately. If a certain box has problems, it will not affect the others and the author can manually fix the problematic boxes one by one.

This paper is organized as follows: Section 2 explains PURE's approach in detail. Section 3 describes our evaluation of PURE. In Section 4, the future work to make PURE more effective is discussed.

2. THE REVERSE ENGINEERING APPROACH

2.1 Background

The content of an HTML document is contained in the `BODY` element. All elements which may appear in `BODY` can be classified into 2 kinds: the "block-level" elements and the "inline-level" elements. Generally, block-level elements may contain inline-level elements and other block-level elements. Inline-level elements may contain only data and other inline-level elements. For example, the elements `P`, `DIV`, `TABLE`, `UL` and `H1` are block-level elements and `B`, `SPAN`, `FONT`, `A` and `IMG` are inline-level elements. During rendering, block-level elements always begin on new lines, but inline-level elements do not [13].

Generally, a Web browser parses the HTML document into a tree structure and then begins rendering based on the tree. The tree structure is often called the HTML DOM (Document Object Model) tree. The W3C's visual formatting model gives the idea on how a browser renders an HTML document based on its DOM tree [14]. Each node in the DOM tree generates one or more boxes on the screen if it is visible. A visible inline-level element may generate more than one box when the text within it spans several lines. A visible block-level element always generates one box because it will begin with a new line. We can see an element's box boundaries by adding the `STYLE` attribute of CSS to specify the border of the box. For example:

```
<P style="border: 1px solid red">
    some text</P>
```

If we specify the border of each element in the document body, we can see that a Web page's appearance is composed of many nested boxes. Modern browsers provide a friendly programming interface for developers to access a page's DOM tree structure and each element's box information (e.g., position, size, source code, etc.). Therefore, it is possible for us to reverse engineer a Web page box by box through the browser's rendering result.

2.2 System Overview

The basic idea of PURE's approach is to reconstruct the Web page's layout using the CSS box model and then fill in the content for each box. When designing page layout with CSS, people commonly use the `DIV` element to represent a box and use the style sheet to position the box of the appropriate size. Since the Web page is composed of many nested boxes, a straightforward way is to use `DIV` to build every box in the original page and then render the new CSS-based page. But this will be messy. For example, the code "`<P>some <B>bold</B> text</P>`" generates two boxes. One box is for element `P` and the other is for element `B`. Wrapping each box with DIV results in the code

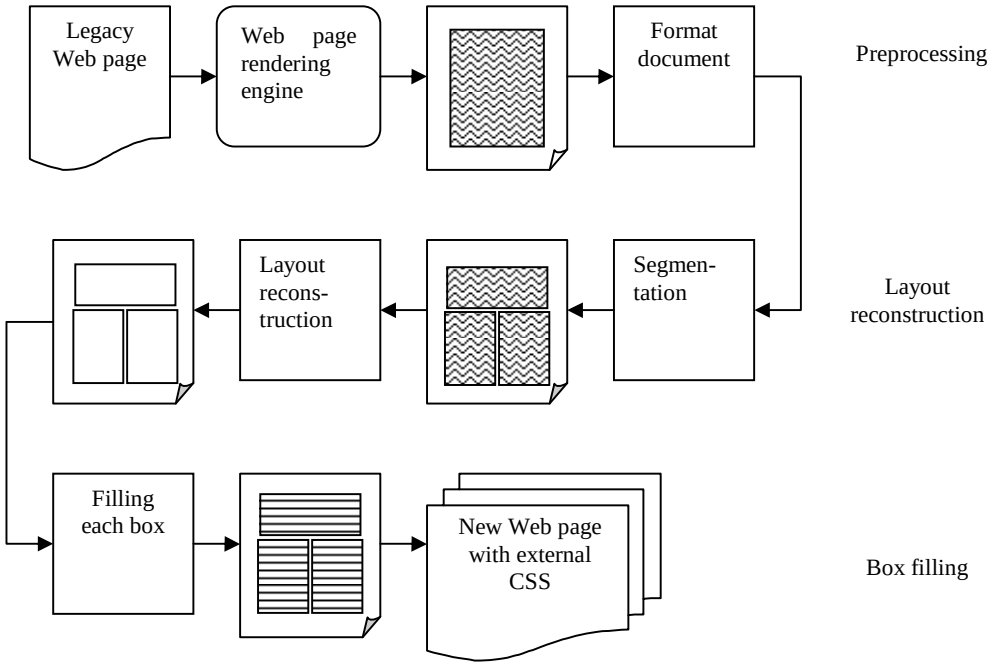

Figure 4. PURE system overview

"<DIV><P>some <DIV><B>bold</B></DIV> text</P></DIV>", which breaks the paragraph. This is not acceptable. Another problem of this simple method is that many redundant boxes will be generated due to the HTML table. For example, the code "<TABLE><TR><TD>some text</TD></TR></TABLE>" generates three boxes because there are three elements here: TABLE, TR, and TD. This will lead to redundant code as "<DIV><DIV><DIV>some text</DIV></DIV></DIV>". Actually, we only need one DIV. From this example, we can also see that HTML table-based layout is not efficient. The above two examples show that we cannot simply use every box in the original page to build the layout of the new page. Instead, we should choose only the necessary boxes. We call these boxes "primary" boxes. The primary boxes cover all the content and are not overlapping with each other. As a result, the primary boxes segment the page into individual rectangles and they will not affect each other. The element which satisfies any of the below conditions is considered a primary box:

- The TD/TH elements within the layout table (layout table cell)
- The leaf block-level elements outside the layout table

A leaf block-level element is a block-level element whose descendant elements in the DOM tree are all inline-level elements. In other words, there is no block-level element inside the leaf block-level element.

In the new Web page, we use CSS positioning techniques [15] to make these primary boxes appear in the same positions as they are rendered by the chosen browser. At the end of this step, the layout reconstruction is finished and the next step is to fill in the content for each primary box. Since each box is corresponding to an element in the DOM tree, the content inside the box is the code of that subtree which has that element as root. Therefore, the approach to fill in the content of each box is to go though its corresponding subtree top-down and generate standard-compliant code.

Figure 4 shows the flow chart of the PURE system. These three major steps are explained further below.

2.3 Preprocessing

This step has two tasks: (1) generate the input Web page's structure by passing it to a popular browser's rendering engine; (2) format the page's structure for later processing. The primary boxes are supposed to cover all the page content. But without formatting the document structure, they may fail to do so. For example, the code "<BODY><P>text one</P> text two <B>text three</B></BODY>" will only generate one primary box, which is the P element. Therefore, the text "text two" and element B will be missing according to the primary box coverage. Our Lemma 1 can solve this problem.

Lemma 1: In the HTML DOM tree, if inline-level elements or text nodes are the siblings of a block-level element, they are implicitly enclosed by the block-level element.

It is easy to proof this lemma. The block-level element always begins with a new line and ends with a line break, which means any element before or after the block-level element will end or begin with a line break. In other words, there is an implicit block-level element containing the inline-level elements or text nodes around a block-level element. As for the above example, the code "<BODY><P>text one </P><P>text two <B>text three</B></P></BODY>" has the same appearance with the

original one. The document formatting process inserts the general block-level element DIV to those places where there are implicit block-level elements. As a result, the above example code will become "<BODY><P>text one </P><DIV>text two <B>text three</B></DIV></BODY>". Figure 5 shows the process. After formatting, two elements will be marked as primary boxes: the elements P and DIV. We can see that they cover the whole content of the page.

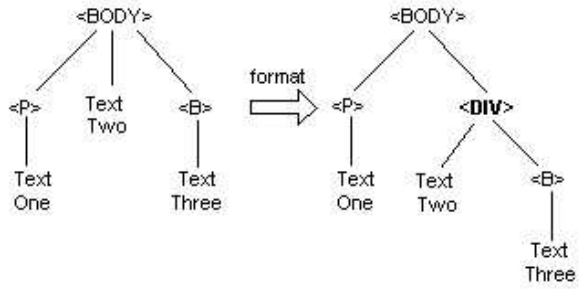

Figure 5. An example of formatting code

2.4 Layout reconstruction

The tasks of this step are: (1) identify the primary boxes; (2) create a new page with the primary boxes in the same position. The primary box is defined as the layout table cell or the leaf block-level element outside the layout table. The primary box identification process goes through the DOM tree top-down and checks every element. If an element is considered a primary box, its descendant elements will not be processed. This guarantees that none of the primary boxes overlaps each other. Note that it is often difficult to distinguish between a layout table and a data table unless we develop an intelligent algorithm which can "understand" the relationship among the content of table cells. This is not the major task of PURE. Instead, we use a simple but effective way. From our observation, when a cell of table contains block-level elements (e.g., P, TABLE, etc), the table is often used for layout. On the other hand, the cells of data tables mostly only contain inline-level elements or text. To identify the leaf block-level element is easy. After the document formatting process, all elements at the same level (i.e., they have the same parent element) are either all block-level or all inline-level. If a block-level element outside the layout table has inline-level elements, it will be made a primary box.

The primary boxes form the layout of the Web page. The next task is to create these boxes in a new Web page with the same positions. As mentioned before, we use DIV element to represent a box and using CSS positioning techniques to control the location of each box. This is a typical CSS box-model based layout design method, which is widely used by the Web developer community. There are three positioning schemes in W3C's CSS recommendation: "normal flow", "floats" and "absolute positioning" [18]. These position schemes are so flexible and powerful that there are always various ways to construct a single layout by using a combination of these schemes. PURE provides two different positioning methods to reconstruct the layout. One is to use absolute positioning and the other is to use a combination of normal flow and floats.

Using the absolute positioning scheme to construct layout is simple and straightforward. PURE generates the boxes one by one and specifies the position and size of each box in the external CSS file. The body of the HTML code is like:

```
<BODY>
<DIV id="FirstBox"> … </DIV>
<DIV id="SecondBox"> … </DIV>
…..
<DIV id="LastBox"> … </DIV>
</BODY>
```

And the external CSS code is like:

```
#FirstBox {position:absolute; top: 10px; left: 10px; width: 800px; height: 100px;}
#SecondBox {position:absolute; top: 110px; left: 10px; width: 200px; height: 400px;}
….
#LastBox {position:absolute; top: 660px; left: 10px; width: 800px; height: 200px;}
```

We can see that the document body structure is a two-level tree and all the primary boxes are in the same level with the same parent element BODY. This document structure may not be identical to the original page. Another problem is that, when developers change a box's size, they need to recalculate the positions of its neighbor boxes and update them in the CSS file. So this approach is not flexible. If the number of primary boxes is not big, this approach is acceptable. But for a complicated page with many primary boxes, this approach may not be suitable.

The second approach, which uses a combination of normal flow and floats schemes to construct the layout, is more flexible than the first one. It does not specify the position of each box. Developers are free to edit an individual box and do not need to update the positioning code of other boxes.

The normal flow is the default scheme in CSS box positioning. In this scheme, boxes flow vertically starting at the top of their containing block, with each one of them placed directly below the preceding one. The effect is illustrated in Figure 6.

```
#box1 {width: 300px; height: 300px;}
#box2 {width: 200px; height: 300px;}
…
<DIV>
  <DIV id="box1">text one</DIV>
  <DIV id="box2">text two</DIV>
</DIV>
```

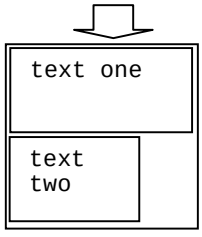

Figure 6. Normal Flow Positioning

The floats scheme works in the horizontal direction. It is achieved by setting an element's `float` style to either `left` or `right`. A box with floats style specified is shifted as far to the right or left of its containing block as possible. If two or more adjacent elements are floated, their tops are positioned on the same line (side by side) if there is sufficient horizontal space to accommodate them. Figure 7 illustrates this effect.

```
#box1   {float:left;   width:   300px;
height: 300px;}
#box2   {float:left;   width:   200px;
height: 300px;}
…
<DIV>
<DIV id="box1">box one</DIV>
<DIV id="box2">box two</DIV>
</DIV>
```

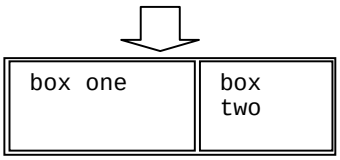

Figure 7. Floats Positioning

With normal flow and floats positioning techniques, we can use nested `DIV` elements to construct any kind of layout of existing Web pages. By inserting sub-boxes with normal flow positioning, a box is split horizontally into rows. By inserting sub-boxes with floats positioning, a box is split vertically into columns. Therefore, we can construct layout like the way an HTML table does it. But this CSS approach is more flexible than the HTML table. In the HTML table, the border lines of columns are aligned across rows and border lines of rows are aligned across columns. In this CSS approach, a box can be freely split into rows and columns without forced alignment.

Considering the process in which Web developers design page layout with nested `DIV` elements and CSS, we designed a top-down recursive algorithm to construct the layout automatically. In the beginning, the Web developers commonly segment a page vertically or horizontally into several major rectangles (boxes). Then they segment each major box into several sub-boxes if necessary. After that, they may segment each sub-box into even smaller boxes. By repeating this process recursively and creating boxes level by level, the desired layout is finally achieved. The PURE's algorithm to construct page layout with nested `DIV` elements and CSS is similar to this process. We assume that all the primary boxes are already at their right places (see example in Figure 8 (a)). At first, we segment the page from top to bottom with boxes. The segmentation method is: we set a horizontal virtual line on the screen. If this line does not cut across any primary box, we consider this line the border line between two boxes. We move the virtual line down and check whether it should be a border line or not. After all the horizontal border lines are detected, the first level segmentation is done and the page is composed of a number of boxes stacked from top to bottom (Figure 8 (b)). Then we try to segment each of these boxes individually. A vertical virtual line inside a box moves from left to right and checks whether it should be a border line. After all the vertical border lines within each box are detected, the second level segmentation is done (Figure 8 (c)). The boxes generated in the second level segmentation are further segmented in a different direction to generate the third level boxes (Figure 8 (d)). This recursive process is repeated until no more boxes can be segmented, which means that the last level boxes are all primary boxes. Thus, the page layout construction is done.

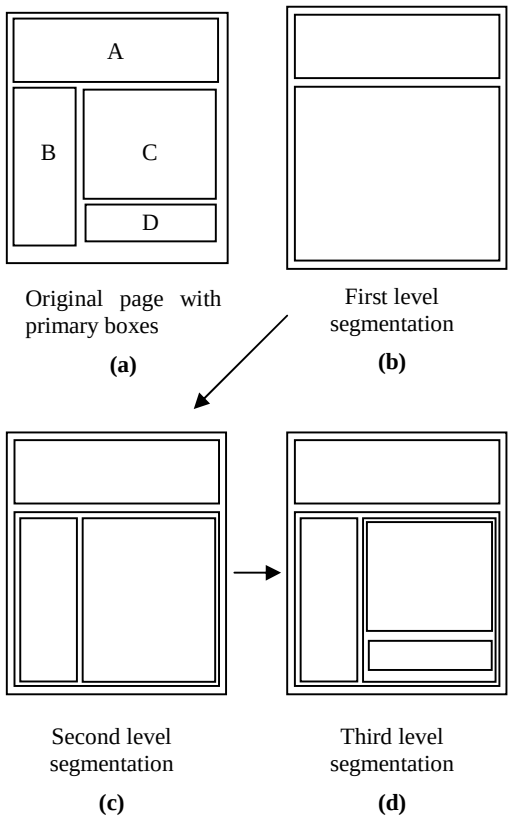

Figure 8. Layout reconstruction algorithm

As for the example in Figure 8, PURE will generate the code below. For simplicity, the corresponding CSS code is not shown here. The code forms the layout of the new Web page and reserves the space for primary boxes. The next step is to fill in the code for each primary box.

```
<BODY>
<!—- first level -->
<DIV>
<!—- code for primary box A -->
</DIV>
<DIV>
  <!—- second level -->
  <DIV>
    <!—- code for primary box B -->
  </DIV>
  <DIV>
    <!—- third level -->
    <DIV>
       <!—- code for primary box C -->
```

```
    </DIV>
    <DIV>
       <!—- code for primary box D -->
    </DIV>
  </DIV>
</DIV>
</BODY>
```

Figure 9 shows the layout of the example page in empty boxes following this process.

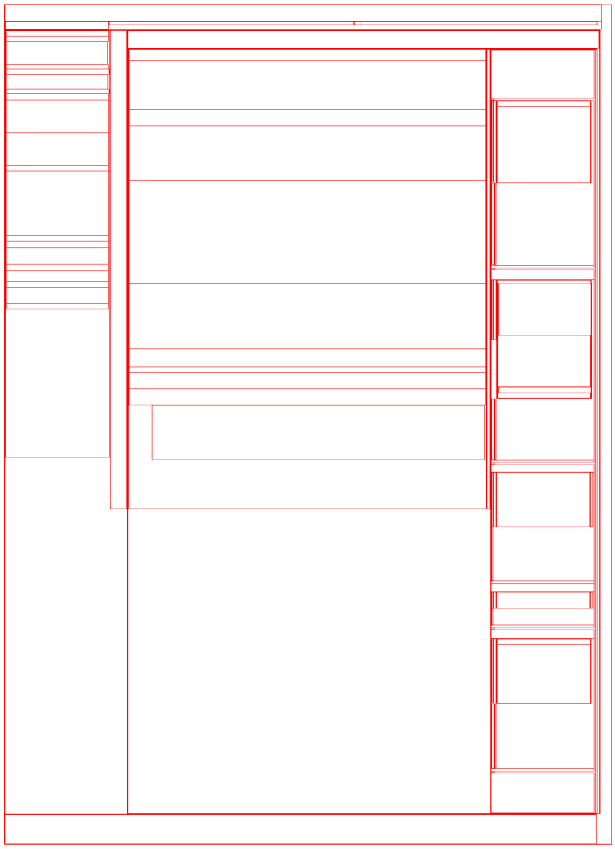

Figure 9. Layout in empty boxes

2.5 Box filling

The last step is to fill in the content of each primary box in the new Web page. Because each primary box is corresponding to a subtree in the DOM tree of the original Web page, PURE goes through the subtree top-down and generates valid code for each node of the tree. Some visual effects (e.g., background color, text alignment and font, etc.) are inherited from ascendant nodes which are outside the primary box. Therefore, the ascendant nodes should be scanned to extract their visual effects and then we use style sheet to format the nodes inside the box. The presentation-related elements inside the box are removed by using style sheet during this process. For example, the HTML elements `FONT`, `B`, `I` and `U` will be replaced by the `SPAN` element and style sheet is used to specify the text appearance. By doing so, all the presentation code in the original Web page is stripped out so that the new HTML document is purely for content and its presentation is purely controlled by CSS. PURE then uses HTML Tidy to make the final clean-up of the generated code for each box.

3. IMPLEMENTATION AND EVALUATION

3.1 Selection of Web page rendering engine

As discussed above, the reverse engineering approach first uses a browser's rendering engine to generate a Web page's structure. Table 1 lists the rendering engines used in well-known browsers and their popularity statistics in October 2005 [4].

Table 1. Rendering engine statistics

Rendering Engine	Used by browsers	Popularity
MSHTML-Modern	Internet Explorer 6.0	82.51%
MSHTML-Legacy	Internet Explorer 4.0/5.0/5.5	3.94%
Gecko	Mozilla, Firefox, Netscape 7.x	9.97%
KHTML	Safari, Konqueror	1.62%
Opera	Opera – all versions	0.83%
Netscape	Netscape 3.x/4.x	0.07%

The rendering engine "MSHTML-Modern", which is used by Microsoft Internet Explorer 6.0, is by far the most popular. It is reasonable to assume that most Web page developers would test their creation with this rendering engine before uploading it to a public Web server. Microsoft also provides sufficient documentation about programming with MSHTML-Modern [6] so that we can obtain enough information about the Web page's structure. For these reasons we use it in PURE. Note that our approach will work equally well with any other browser as long as the structure information is available.

In order to embed MSHTML-Modern in PURE, the software is developed on Windows XP in C++.

3.2 Graphic user interface

Since we do not expect PURE to be perfect in fixing invalid Web pages automatically, it is designed to have a graphical user interface (GUI) so that the user may fix some problems interactively whenever needed. Through PURE's GUI the user may adjust the resulting layout conveniently. PURE uses HTML Tidy to make sure the code used to fill in the boxes are valid HTML code. However, the user is given the opportunity to change that code manually if Tidy's result is not suitable. Figure 10 shows an example of a PURE pop-up window which the user may use to fix the HTML code easily.

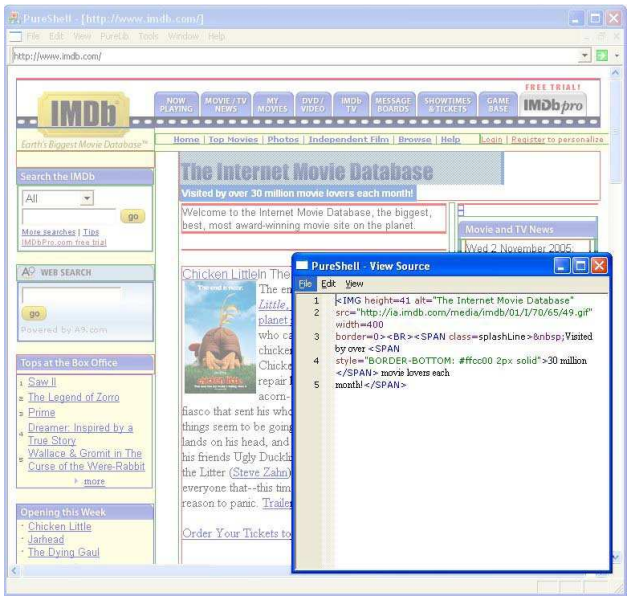

Figure 10. Pop-up window for code fixing.

3.3 Evaluation

Our evaluation is to test whether the PURE tool can successfully reconstruct existing Web pages automatically. The normal flow and floats positioning technique is used to generate the layout because it is much more flexible than absolute positioning. The testing Web pages used are the homepages of the top 500 websites [1]. These top websites are mostly portal websites and their homepages are among the most complicated cases. Also, there are different languages used in the top 500 websites (242 English sites, 121 Chinese sites, 45 Japanese sites, etc.). We can also test PURE's robustness on processing different languages. The similarity between the original Web page and PURE-generated Web page is used as the subjective evaluation criteria by the authors. A rating of 100% means that the PURE-generated Web page appears the same as the original. A rating of 90% (such as the IMDB website example in Figure 3) means that the PURE-generated Web page appears "almost" the same as the original except with some minor differences. A rating of 80% means that the PURE-generated Web page is similar to the original, but one or two places are noticeably wrong. A little manual editing through PURE's GUI can help these 80% pages to appear the same as the original one. A rating of below 80% is considered as a failure of PURE.

In order to make the testing Web pages less relying on their Web servers' special features, we use a Web grabber to download the Web page to a local machine and test PURE with this saved Web page. Only 440 testing pages (of the top 500) can be successfully opened in the local machine. The evaluation result is shown in Table 2.

Table 2. Evaluation result of PURE

Similarity	100 %	90%	80%	>=80% (Success)	< 80% (Failure)
Number	70	51	103	224	216
Rate	16%	12%	23%	51%	49%

The results show that PURE is able to successfully reconstruct over 50% of the Web pages collected from the top 500 (or rather, 440 of the top 500) websites. We studied the failure cases and found that most of the failures are caused by the inconsistent page structure between the one generated by MSHTML-Modern and W3C's definition of the DOM tree.

4. DISCUSSION

Today, people have realized that most of existing Web pages are not compliant with Web standards, which hinders the accessibility and viability of the Web. Transforming those legacy Web pages to be standard-compliant is necessary but will require a lot of effort. Our PURE tool is developed to help Web developers to do the transformation automatically. Because some legacy Web pages are very complicated and the browser may not generate the page structures according to W3C's definition of the DOM tree, PURE may not work well automatically in some cases. But with some human interaction, PURE can be effective for most cases. Furthermore, the PURE tool will be released as an open source and free software so that the community can help improve it. We believe that PURE will evolve into a useful tool for Web developers to transform legacy Web pages. With additional work PURE may be made an option for Web page authors to pass the code through before leaving the authoring tool used for the page's creation.

Web page reverse engineering can be applied to many other applications. One possible application is to transform existing Web pages to make them compatible with mobile devices. Today, mobile Web access suffers from interoperability and usability problems that make the Web difficult to use for most mobile phone subscribers [19]. One well-known problem is that traditional Web pages do not provide a good browsing experience on mobile devices due to their small screen size. This problem may be solved by using a new presentation scheme (style sheet) with which the browsers on mobile devices will use to display the Web page nicely. Some popular websites now provide a version designed specially for mobile devices such as http://www.google.com/pda and http://wap.oa.yahoo.com/ . But it would be difficult to keep the content consistent with the original Web page unless there is a tool to generate a mobile version automatically from the traditional website.

The reverse engineering approach introduced in PURE can be adopted to help Web developers build Web pages that are compatible with mobile devices based on existing ones. That is: (1) segment the original Web page into primary boxes; (2) remove some primary boxes if necessary (e.g., large images or navigation links on the top of the page [20]); (3) rearrange the placement of primary boxes to fit the mobile device's screen. Therefore our reverse engineering approach will become a valuable foundation for research work on Web standards and Mobile Web.

5. PROTOTYPE

Rather than providing additional evaluation data, which are necessarily subjective, a PURE prototype is now available for public testing and comments at http://webproject.cs.ust.hk:8004/. It is free and open source software and published under General Public License (GPL) [3]. Readers are welcome to improve it or develop applications based on it.

6. ACKNOWLEDGMENTS

This research was supported in part by Sino Software Research Institute grant SSRI01/02.EG14, "W3C Office" and by Research Grant Council grant AOE/E-01/99, "Information Technology for a 21st Century Hong Kong". We want to thank Dan Hong, Yongzhen Zhuang and Shan Chen for their valuable contributions to this work.

7. REFERENCES

[1] Alexa.com. *"Top 500 sites."* http://www.alexa.com/site/ds/top_500

[2] European Computer Manufacturers Association (ECMA). "Standard ECMA-262: ECMAScript Language Specification". http://www.ecma-international.org/publications/standards/Ecma-262.htm

[3] GNU Foundation. "GNU General Public License". http://www.gnu.org/copyleft/gpl.html

[4] John Haller. "Browser Rendering Engine Statistics". http://johnhaller.com/jh/useful_stuff/browser_statistics

[5] MaxDesign.com . "The benefits of Web Standards to your visitors, your clients and you!". http://www.maxdesign.com.au/presentation/benefits

[6] Microsoft Corporation. "Programming and Reusing the Browser". http://msdn.microsoft.com/workshop/browser/prog_browser_node_entry.asp

[7] François Nonnenmacher. "Web Standards for business". http://www.webstandards.org/learn/reference/web_standards_for_business.html. 2003

[8] Dave Raggett. "Clean up your Web pages with HTML TIDY". http://www.w3.org/People/Raggett/tidy

[9] Chen Shan, Hong Dan, Vincent Shen. "An Experimental Study on Validation Problems with Existing HTML Webpages ". *Proceedings of International Conference on Internet Computing (ICOMP'05)*, Las Vegas, 2005. pp. 373-379.

[10] The Web Standard Project. http://www.webstandards.org

[11] Jeffrey Veen. "The Business Value of Web Standards". http://www.adaptivepath.com/publications/essays/archives/000266.php

[12] World Wide Web Consortium (W3C). "Web Content Accessibility Guidelines". http://www.w3.org/TR/WAI-WEBCONTENT

[13] World Wide Web Consortium (W3C). "The global structure of an HTML document". http://www.w3.org/TR/REC-html40-971218/struct/global.html

[14] World Wide Web Consortium (W3C). "Visual formatting model". http://www.w3.org/TR/REC-CSS2/visuren.html

[15] World Wide Web Consortium (W3C). Positioning HTML Elements with Cascading Style Sheets http://www.w3.org/TR/1999/WD-positioning-19990902

[16] World Wide Web Consortium (W3C). "HTML 4.01 Specification". http://www.w3.org/TR/REC-html40

[17] World Wide Web Consortium (W3C). "What is the Document Object Model?". http://www.w3.org/TR/DOM-Level-2-Core/introduction.html

[18] World Wide Web Consortium (W3C). "Positioning schemes". http://www.w3.org/TR/CSS21/visuren.html#positioning-scheme

[19] World Wide Web Consortium (W3C). Mobile Web Initiative. http://www.w3.org/Mobile

[20] World Wide Web Consortium (W3C). "Mobile Web Best Practices 1.0". http://www.w3.org/TR/2005/WD-mobile-bp-20051017

[21] Jeffrey Zeldman. *"Designing with web standards"*. New Riders: Berkeley, 2003. 456 pages.