



SCHOOL OF  
COMPUTER SCIENCE

Information  
Management Group

HCW— WIMWAT Technical Report 1, May 2009

## Identifying Web Widgets

Technical Reports

**Alex Q. Chen and Simon Harper**

Human Centred Web Lab  
School of Computer Science  
University of Manchester  
UK

The Web 2.0 concepts encourage presented content to be updated dynamically, without the need for Web pages to be reloaded. However, these concepts require assistive technologies to adapt the way that they interact with Web pages. This report provides an insight into the problem, and introduces a process to identify and modify the problem at the developer's end, so that Web widgets that produces inaccessible micro-content can be modified into an accessible form. The proposed process is divided into three stages, and this report details our investigation of the first stage; the identification stage. An evaluation of the methods used in our identifying process was conducted for two Web widgets on twenty Websites. It demonstrated that our approach is promising by achieving a 100% successful detection rate, but there were a few false positive detections. Refinements to our approach are suggested to overcome the issues raised from this investigation. We believe that we have proven that this method is feasible, and the suggested future work should be pursued.

HCW

Human Centred Web

## **WIMWAT**

The aim of the Widget Identification and Modification for Web 2.0 Access Technologies (WIMWAT) project is to comprehend Web page's source code, so that Web widgets that produces inaccessible content can be identified, in order to modify them into an accessible form during development. The WIMWAT Web ages may be found at <http://hgw.cs.manchester.ac.uk/research/>.

### **WIMWAT Reports**

This report is in the series of HCW WIMWAT technical reports. Other reports in this series may be found in our data repository, at <http://hgw-eprints.cs.man.ac.uk/view/subjects/wimwat.html>. Reports from other Human Centred Web projects are also available at <http://hgw-eprints.cs.manchester.ac.uk/>.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Technical Background</b>	<b>2</b>
<b>3</b>	<b>Experiment Setup</b>	<b>5</b>
3.1	Scripting Language . . . . .	5
3.2	Experimental Data Set . . . . .	5
<b>4</b>	<b>Auto Suggest List (ASL) Widget</b>	<b>6</b>
4.1	The “tell” Signs . . . . .	6
4.2	Assumptions and Rules . . . . .	11
4.3	Limitations . . . . .	11
4.4	Results and Discussions . . . . .	12
<b>5</b>	<b>Carousel Widget</b>	<b>14</b>
5.1	Slide Show Widget . . . . .	15
5.2	Module Tabs Widget . . . . .	16
5.3	The “tell” Signs . . . . .	17
5.4	Assumptions and Rules . . . . .	20
5.5	Limitations . . . . .	20
5.6	Results and Discussions . . . . .	21
<b>6</b>	<b>Future Work</b>	<b>24</b>
6.1	Time Out “tell” Sign . . . . .	24
6.2	Static Object Model “tell” Sign . . . . .	24
6.3	Annotating the Source Code . . . . .	26
6.4	More Comprehensive Evaluation . . . . .	26
<b>7</b>	<b>Conclusion</b>	<b>26</b>

---

**Human Centred Web Lab**  
School of Computer Science  
University of Manchester  
Kilburn Building  
Oxford Road  
Manchester  
M13 9PL  
UK

**Corresponding author:**  
Alex Qiang Chen  
tel: +44 (0) 161 275 7821  
chenqa@cs.man.ac.uk  
<http://homepages.cs.manchester.ac.uk/~chenqa>

## 1 Introduction

The second generation of Web development and design (Web 2.0) concepts brought many new advancements to the Web development community. These concepts encourage Web applications to source data and remix them from multiple sources, and to utilize the advantages of that platform to deliver rich user experiences [12]. Web pages that uses the Web 2.0 concepts present their content such that they update in small chunks called micro-content, rather than reloading the entire Web page [4]. This way of presenting information will affect the way that assistive technologies such as screen readers will interact with Web pages.

The quest to provide solutions to smoothen the adaptation process for assistive technologies, so that these technologies can merge into the Web 2.0 concepts has led us to an identifying and modifying process. This approach will identify the different types of Web widgets used on a Web page, so that widgets that produce inaccessible content can be identified, and corrected at the developer's end.

Our approach requires techniques used by Web mining, pattern recognition, and understanding the semantics of different elements that form the Web widget's interface. Since the Web is a heterogeneous set of technologies, recommendations, and guidelines, some of the conventional techniques will need to be modified, or even a fusion of these techniques will be required in order to achieve what it is meant to do. In order to encapsulate the different techniques employed, "tell" signs<sup>1</sup> are employed as clues within the code. When they are combined under certain conditions, these "tell" signs will form the identity of a specific Web widget.

Flexibility is incorporated in the detection methods used by our identification process. This agility allows our methods to cover a range of programming styles and practices accustomed by the developers. This process uses code comprehension alongside with a legitimate set of rules to govern the "tell" signs, so that a Web widget can be discovered within a Web page.

An evaluation was conducted to detect the usage of two Web widgets from twenty Websites. These Websites were selected from Alexa's Global 500 sites on the Web list<sup>2</sup>. The widgets chosen for this evaluation were based on the way they responded to a request when a service is requested by the users. In this case, through these widgets, content presented on a Web page will be affected via dynamic micro-content updating. These widgets are the Auto Suggest List (ASL) and the Carousel widgets.

The results from our evaluations demonstrated that our detection methodology is promising, and it has the capability to do the job. The evaluation of the methods used by both Web widgets achieved a 100% successful detection rate. However, a few false positive detection occurrences were noticed from the results. These glitches suggested that refinements are required to improve the reliability of our methods. Suggestions to overcome these issues are presented in the future work section.

### Synopsis

This report is structured as follows:

---

<sup>1</sup>"tell" signs are clues within the code that gives the "smell" and characteristics of a Web widget.

<sup>2</sup><http://www.alexa.com/topsites>

**Section 2: Technical Background** defines the terms used in this report, and the knowledge required to understand this report.

**Section 3: Experiment Setup** explains the scripting language used to evaluate our approach, and the assumptions and methods used to prepare our data set for the experiments conducted in this study.

**Section 4: Auto Suggest List (ASL) Widgets** describes the functionality of the ASL widget and how it affects the displayed content on a Web page. Next the assumptions, limitations, and methods used to evaluate our approach of the ASL widget are covered. Finally discussions on the results from our evaluation are presented to conclude on the ASL widget detection experiment.

**Section 5: Carousel Widgets** describes the functionality of the Carousel widget. The assumptions, limitations, and methods used to evaluate our approach of the Carousel widget were covered. Finally discussions of the results from our evaluation are covered.

**Section 6: Future Work** discusses the possible methods that can help to provide solutions to the issues raised in section 4 and 5. Suggestions to provide a more comprehensive investigation for our approach are also presented.

**Section 7: Conclusion** summarises the results and findings for this study, and suggests further work.

## 2 Technical Background

During the course of this report, a number of terms will be used to explain our approach and to report the results. This section defines some of the common terms used, and the knowledge required to understand this report.

### Web Pages

The meaning of a Web page can vary but, in this report it refers solely to the code that was transmitted to the client-side that a user-agent examines. This is code containing contents such as HyperText Markup Language (HTML), Extensible HTML (XHTML), JavaScript and Cascading Style Sheets (CSS). It does not include any server-side code or multimedia files code. This code is the bare minimum that most popular user-agents should be able to interpret.

### Multimedia Files

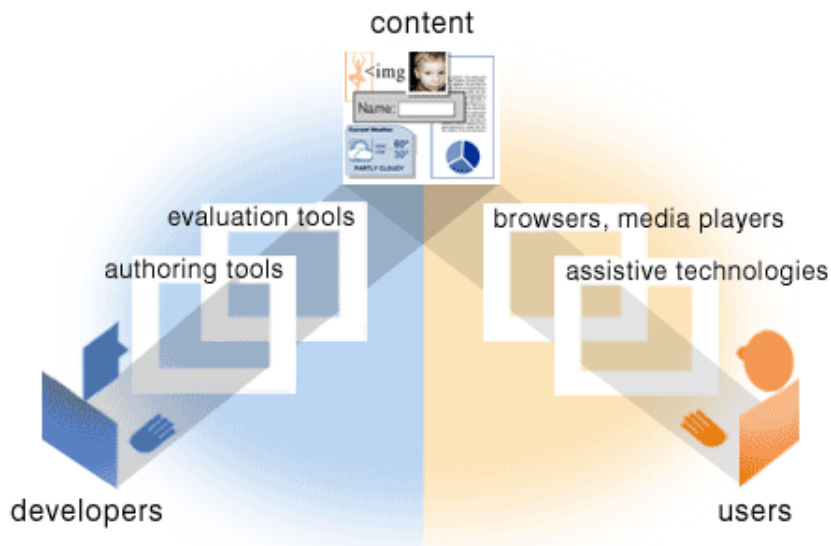
Multimedia files are file formats that can deliver interactive, motion images, and audio over the Web. These are file formats for movies, audio, and animations. Some examples of such files are .swf (Flash), .mov, and .mp3. These kinds of file formats will not be covered in this report.

## Web Widgets

Web widgets consist of code that can be embedded within a Web page that causes the content to change. They are triggered by events either through a timer or from a reaction by the user. Examples of Web widgets are the ASL widget and the Carousel widget.

## Assistive Technologies

Most users of the Web interact with it via a keyboard or a mouse, and receive the output visually on a screen. Assistive technologies were developed to provide access to people who cannot use conventional methods. Highlighted by Edwards, the Web was developed mainly for the visual medium [7]. This causes a real problem for people with visual disabilities. Figure 1 shows how the different components falls into place so that the Web can be accessible.



**Figure 1:** The different components required to develop a Web page, and accessing it. Figure courtesy of <http://www.w3.org/WAI/intro/relate.png>

## Design Patterns

A design pattern is a solution to a problem faced by developers repeatedly over time. Depending on the purpose and the experience of the user, each pattern can be interpreted differently. Gamma et al. discussed in [9] the importance of identifying the abstractions of the design patterns. It is a key to make the design flexible and reusable. However, it can vary hugely depending on the developer's experience, and the purpose of the development. According to Alexander, patterns are expressed as rules of thumb, and can be combined and recombined to suit the different needs when designing [1].

Design patterns are normally applied during the development stage of an application life cycle, but we are doing it in the reverse order, trying to detect a design pattern from the application source code after it is developed. Due to the organisation of the Websites, during development, design patterns may be broken down into many parts to form objects, or may be it was done in a certain fashion due to the way the developers decided to organised it. Therefore, determining the granularity of the design patterns is made more difficult by these issues. As mentioned by Bosch, the issues with breaking down the design patterns, and reorganising it to suit the structure of an application, is that this could lead to traceability problems in the later part of the application life cycle [3].

### **Reverse Engineering**

Design patterns are applied during the development or maintenance phase of a Website life cycle. Relying on only the client Web page's source code makes it difficult, sometimes impossible to recreate the intent of the code. Instead, we reverse engineer the widgets, searching for instances within the source code that forms an identity of a possible pattern that describes the widget.

Each widget often have a number of identities and characteristics that makes it unique. We refer to the individual identities or characteristics as “tell” signs in this report. So if a Web page contains all the “tell” signs of a specific widget, we can assumed that the widget exist on the Web page.

### **Remote Scripting**

Remote scripting is a call-response-reload model used in Hypertext Transfer Protocol (HTTP) transactions [2]. It allows the client and server to exchange information, and from the result returned by the server, the client can make necessary changes to the content without reloading the Web page. A commonly known remote scripting model is the Asynchronous JavaScript and XML (AJAX). It allows the exchangeable content to be formatted as either Extensible Markup Language (XML), HTML, plain text, or JavaScript Object Notation (JSON) [11]. As discussed by Apple Developer Connection, remotng scripting can also be implemented with iFrames [2]. This method allows a Web page to contain a Web document within it. Similar to frames, the Web page does not need to reload whenever the Web document in the iFrame navigates from one Web document to another.

### **Client-Side Scripting**

Client-Side scripting language is a class of computer programs that can manipulate and display content returned from the server. It enables Web developers to respond to user events by changing the way the content is displayed [5]. Often client-side scripts are embedded within a hypertext document over the Web, however it is also possible to include client-side scripts that were coded in files separate from the hypertext document.

JavaScript will be the only client-side scripting language covered in this study. This form of client-side scripting language is lightweight, and can be interpreted as a

programming language with object-oriented capabilities. The JavaScript language's syntax resembles popular programming languages such as C, C++, and Java. It can be used to interact with the user, control the Web browser, and dynamically create content and elements within the hypertext document. Since JavaScript is an interpreted language like Perl, it also includes capabilities such as regular expression and array-handling features [8].

### 3 Experiment Setup

Two experiments using the same experimental data set were done to evaluate our approach. This section explains the essential components, methodologies, and assumptions made when preparing the experiments.

#### 3.1 Scripting Language

Our evaluations were written in the PHP scripting language because of the availability of its application program interfaces (APIs), types of regular expression syntax, and it is freely available. Regular expressions play a crucial role when searching for clues within a Web page's source code, so that "tell" signs can be discovered. Due to the strength of the tools available to match complex patterns within a string, Perl syntax was chosen for scripting the regular expressions.

#### 3.2 Experimental Data Set

Our data set consisted of twenty Websites selected from Alexa's global top five hundred sites on the Web list<sup>3</sup>. When choosing our set of data, a few considerations were made. One of these was to use a range of popular Websites so that it will give a good representation of the broader Web. As discussed in our earlier study on Web Evolution, the top twenty Websites have proven to give a good representation of the broader Web [6]. Hence the twenty Websites were chosen from the Alexa's global top five hundred Websites list using the following rules.

1. No repetition of a Website's domain was allowed. For instance, google.com and google.com.br, because google.com.br is a sub-domain of google.com, thus this will be considered as a repetition.
2. If a repetition exist, it will be ignored and the next domain down the list will be examined from the first point of this list again.
3. All Websites chosen must be accessible, and broken links will be ignored.

The above requisites will be repeated for all Websites from the Alexa's global top five hundred Websites list, until twenty Websites were selected for our experimental data set.

---

<sup>3</sup><http://www.alexa.com/topsites/global> — The Alexa's global top five hundred sites on the Web list is compiled from the lists of Websites ordered by Alexa's Traffic Rank.

Secondly, to ensure repeatability, the same set of data collected on 19 February 2009 was used when conducting both experiments. This will enable us to freeze any changes to the Websites during the course of the experiments.

Thirdly, Websites can tailor their design specifically to a user-agent. Mozilla's Firefox<sup>4</sup> was chosen as the user-agent when capturing our data set. This will enable us to capture Websites with Web pages that complies to one of the industry's leading user-agents.

## 4 Auto Suggest List (ASL) Widget

The ASL widget is a presentation model to convey possible suggestions to the user when he/she is completing a text box field from a form on a Web page. On most popular Web browsers, an auto complete feature, similar to the ASL widget, also provides suggestion to the user when completing a text box field from a form. However, the auto complete feature suggest only related entries that are previously submitted on that computer. Therefore, the ASL widget is not only a replacement for the auto complete feature, but it also extends its capability, by providing suggested texts relating to the purpose of the text box field, e.g., the names of airports, and the names of train stations. This widget provides instant help and clues for users when filling in the text box field. By providing such a feature to the user when filling in a text box field, it not only assistance to the users when completing the Web form, but it also ensures that the text box field's content get filled in more accurately.

An example of the ASL widget in use on a Website is shown in the non-shaded region in figure 2. This widget will update its suggested items whenever the user enters or removes a character from the text box field. The user can navigate the list of suggested items using either a mouse, or by using the up and down keys on the keyboard. The client interacts with the server to provide the users with the relevant suggested items. This interaction process is repeated every time the users changes the content in the text box field.

Content accessibility is particularly important for this widget. Since a lot of content changes when the user interact with the widget, the content fed back is quite important to the user. It may even affect the accuracy of the form completed.

### 4.1 The "tell" Signs

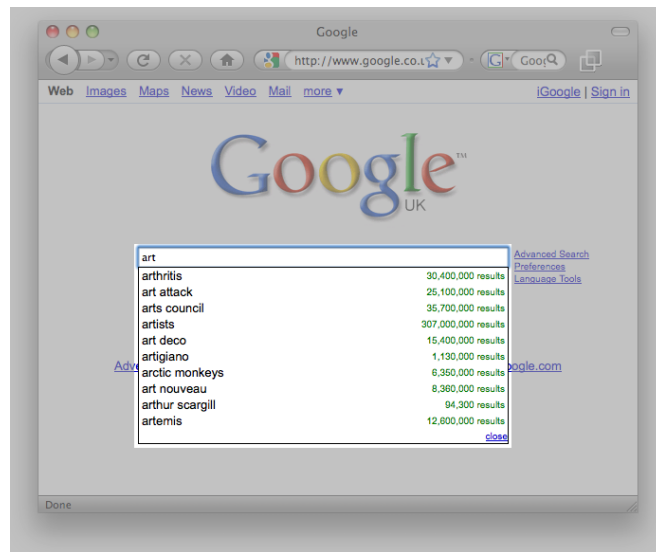
A few "tell" signs were introduced so that it will be possible to identify the ASL widget from the source code. Extensive investigations were conducted on Web pages with ASL widgets, Yahoo! Developer Network's design pattern library<sup>5</sup>, and Welie's pattern library<sup>6</sup>. Additional definitions for the objects within the widget are required to be introduced or redefined to match the granularity of our patterns. An example of this is the form buttons. Unlike conventional standalone applications, a button object can be in a form of a image, a string of texts, or a button.

---

<sup>4</sup><http://www.mozilla.com/firefox/>

<sup>5</sup><http://developer.yahoo.com/ypatterns/>

<sup>6</sup><http://www.welie.com/patterns/>



**Figure 2:** An example of ASL widget (non-shade region) in action on Google.co.uk

Five “tell” signs were employed to spot if an ASL widget existed within a Web page. Each of the “tell” signs will require a sequence of conditional clues when comprehending the code before it can be determined. The five “tell” signs will be covered individually in the next few sub sections.

### Auto Complete Off “tell” Sign (ts1)

The Auto Complete Off “tell” sign examines the Web page for clues that the Web browser’s Auto Complete feature is disabled. This is done by either setting it to ‘off’ in the HTML code, or by programming it in JavaScript. The following regular expression is used to search within the JavaScript code. It searches for signs that the JavaScript code was programmed to set the Auto Complete feature’s attribute to off.

```
/[-_\.a-z0-9]+\setAttribute(\s)*\((\s)*(\\"|\')autocomplete(\\"|\'),
(\s)*(\\"|\')off(\\"|\')(\s)*\)(;)?/i
```

If the above regular expression could not be found, then an attempt will be made to search in the HTML code. The following regular expression is used to search for the setting of the Auto Complete feature within the HTML code.

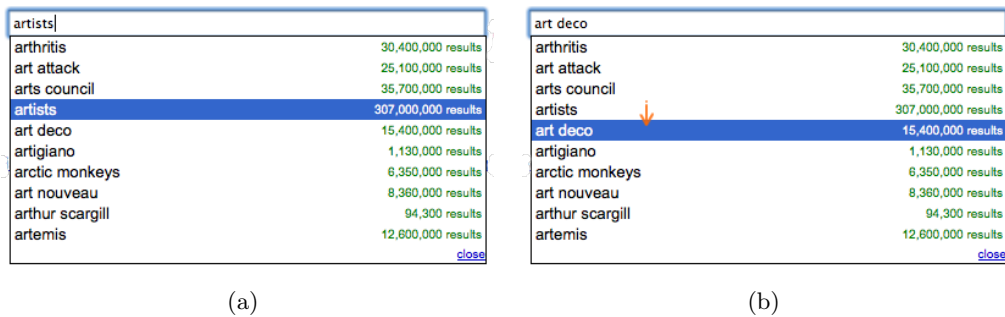
```
/autocomplete(\s)*=(\s)*(\\"|\')?off(\\"|\')?/i
```

### Up and Down Keys “tell” Sign (ts2)

The Up and Down Keys “tell” sign checks if the Web page attempts to poll for the up and down keys. Commonly the users of the ASL widget can traverse through



**Figure 3:** To move the selection up as seen from subfigures (a) to (b), press the up arrow key (key code 38)



**Figure 4:** To move the selection down as seen from subfigures (a) to (b), press the down arrow key (key code 40)

the suggested list of items using the up (key code 38) and down (key code 40) keys on the keyboard (see figure 3 and figure 4 for illustration respectively).

From our investigations over the Websites with ASL widgets and the patterns libraries, two methods are commonly found when developing the ASL widget; either using existing APIs or custom-made. Hence, a few methods can be used when polling for the up and down keys. In the event when both key codes cannot be detected, the system will check for the “keyup” and “keydown” keywords within the JavaScript code. This is because some APIs may provide this feature as a replacement for the key code polling. To ensure that the polling for the up and down keys are intended for the ASL widget, the event that triggers this function must be examined to ascertain the relationship between the hypertext elements, and the JavaScript code.

This “tell” sign is broken into a few parts, to cater for the diverse styles and practices of programming, and the different APIs used. The following two regular expressions are applied to detect if the code poll for certain keys. The first regular searches whether the ‘keyCode’ function in JavaScript was called. If that regular expression is true, the second regular expression will be followed to check whether key code 38 and 40 were polled by the ‘keyCode’ function.

```
/* To check if keyCode function was called */
/[_\.a-z0-9]+\.\.keyCode/i
```

```
/* To check for the correct key codes were polled */
/(38|40)/i
```

However, if the Yahoo API<sup>5</sup> is used, the following regular expression will be used for detecting the polling of the up and down keys instead of searching for key codes 38 or 40.

```
/YAHOO.util.Event.addListener\([-_\.\a-z0-9\,]+,(\s)*\"
  (keyup|keydown)\"/i
```

Next the events that trigger the functions or API must be detected, so that we can relate sections of the JavaScript code that form the ASL engine with the hypertext elements. This was done by using either of the following regular expressions.

```
/\.\focus\(\)/i
```

```
/\.\event/i
```

In the event if all the above checks fail, another attempt to search for keywords such as ‘keyup’ and ‘keydown’ will be made within the JavaScript code using the following regular expression.

```
/\"(keyup|keydown)\"/i
```

This attempt is made, because tailored APIs may provide an inbuilt key code detection feature, and keywords may be used as variables to refer to or trigger these features.

### Updates “tell” Sign (ts3)

The Updates “tell” sign detects if the code attempts to append the document object model (DOM) node for a table, list or anchor element during the time a user spent on the Web page. When the list of suggested items is updated, the DOM needs to be updated too, so that Websites can format their suggested results either in a list or table. Some may even use an anchor element for each item in the list of suggested items. To reformat the existing Hypertext document, this is done by using the JavaScript’s “appendChild” function. Due to the available options for listing these item, this “tell” sign is split into two parts. The first part uses regular expressions to search if either a table, list or anchor element was used in the JavaScript to append the child.

```
/(\"|\')table(\"|\')/i      /* table */
```

```
/(\"|\')td(\"|\')/i        /* table */
```

```
/(\"|\')li(\"|\')/i        /* list */
```

```
/(\"|\')a(\"|\')/i         /* anchor */
```

The second part ensures that the “appendChild” function in JavaScript is used with either of the detected elements found in part one. This part uses the following regular expression to search for this clue.

```
/([-_\.a-z0-9])*\.appendChild(\s)*\(/i
```

#### **Clear List “tell” Sign (ts4)**

The Clear List “tell” sign searches for clues within the JavaScript code that will clear the previous list of suggested items. Previous results for the suggested list will need to be cleared before new results can be updated to the suggested list. This method is commonly used by developers when updating their list of suggestions.

Developers can either clear the content, or remove the nodes within the list. To clear the list, one can use an empty string to clear the content, but this method does not tell us much about the intent of the developers. So this method will not be searched as it will not be possible to know the original intention. Thus, only the following regular expression will be used to search for clues that the node of the previously suggested list is removed.

```
/([-_\.a-z0-9]+\).removeChild(\s)*\(/i
```

#### **Remote Scripting “tell” Sign (ts5)**

The Remote Scripting “tell” sign searches for clues that remote scripting is used by the widget to complete its task. During the ASL widget’s service process, it requires the server to return a list of suggestions related to the content in the text box field, so that micro-content can be updated. Remote scripting can be requested using a few methods such as AJAX, or iFrames.

We began by searching for clues that iFrames were used. To search for this, two clues must be present. 1) The iFrame tag must be used in the HTML code. This can be done either by hard coding, or created dynamically by scripting languages. Hence, a generic regular expression was used to search for these clues.

```
/iframe/i
```

2) CSS code must be used if iFrames were employed. Only by this means will the results from the ASL widget to float over the existing content. To do this in CSS, the ‘position’ attribute must be set to ‘absolute’. This is searched using either of the following regular expressions.

```
/position(\s)*=(\s)*(\\"|\')absolute(\\"|\')/i
```

```
/position(\s)*:(\s)*absolute/i
```

If iFrame is not found to be used by the ASL widget, then the HttpRequest method is searched instead. To determine whether the widget uses the HttpRequest method, either of the following regular expressions must be true before we can determine that this method of remote scripting was used.

`/\ XMLHTTP /i`

`/XMLHttpRequest\(\)/i`

`/HttpRequest/i`

## 4.2 Assumptions and Rules

A number of assumptions were made when designing the “tell” signs, and during the process of determining whether the ASL widget is used by the Web page. Firstly, not all the “tell” signs need to be satisfied before an ASL widget can be assumed to exist on a Web page. Two rules were created to govern the conditions when determining if an ASL widget existed on a Web page.

**Rule 1:** The first three “tell” signs ( $ts1$ ,  $ts2$ ,  $ts3$ ) must be true.

**Rule 2:** Either “tell” sign 4 or 5 (or both) must be true.

Described in equation 1, both of the above rules must be satisfied before an ASL widget can be assumed to exist on a Web page. The second rule is a conditional rule, this is designed to cater for the variations of programming styles and practices from one Website to another.

$$ts1 \wedge ts2 \wedge ts3 \wedge (ts4 \vee ts5), \quad (1)$$

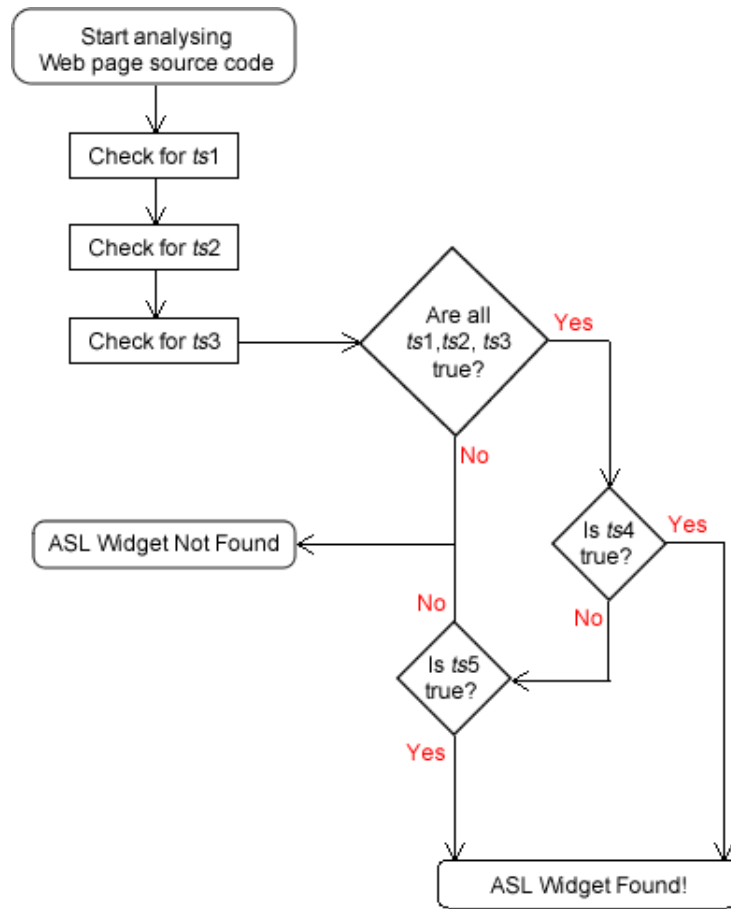
where  $ts1$  is the Auto Complete Off “tell” sign,  $ts2$  is the Up and Down Keys “tell” sign,  $ts3$  is the Updates “tell” sign,  $ts4$  is the Clear List “tell” sign, and  $ts5$  is the Remote Scripting “tell” sign. Using this set of rules, the Web page’s source code will be comprehended according to the flow chart illustrated in figure 5. The code is structured to examine  $ts1$  to  $ts3$  first, before it will decide whether to proceed to  $ts4$  then  $ts5$ . This method of coding was used to make the code leaner, and more memory efficient.

Secondly, we will only investigate ASL widgets written in JavaScript. This assumption was made based on a recent study conducted by us. It showed that VBScript (a competing client-side scripting language with JavaScript) was poorly adopted over the last 10 years [6], and JavaScript was the preferred client-side scripting language by most Websites. ASL widgets written in other technologies such as Flash are also not investigated in this study.

## 4.3 Limitations

This evaluation is designed to analyse documents formatted in Hypertext, CSS, and JavaScript only. ASL widgets that are written in Flash, Shockwave, VBScripts, or any other formats will not be covered.

A widget can be coded in a variety of styles and practices. Our methodologies cover a range of styles and its variants. We detect for instances of a pattern rather than the pattern itself, so that flexibility can be incorporated into our detection



**Figure 5:** ASL widget detection’s “tell” sign process flow chart

methods. However, it does not cover all variations of it, and all practices used to develop the widget. For example, if the set of code, used for controlling the behavior of the ASL widget is generated by another set of client-side scripting code, than the set of code that will control the ASL widget will slip through our detection methods.

#### 4.4 Results and Discussions

The results for detecting the ASL widget experiment are tabulated in table 1. The results on the table can be interpreted as ‘1’ symbolised a true, ‘0’ symbolised a false, and ‘X’ symbolised ignored for some reason during the investigation. In the first column, ‘Manual detection’ this was done manually by a human being. If at least one ASL widget is present on the Website, it will be given a ‘1’ in this column, and a ‘0’ if no ASL is found. The second column, ‘Auto detection’ employs the rules discussed in section 4.2 to determine if a ASL widget exist on the Web page from the “tell” signs results. Following that, the evaluation results for the individual “tell” signs are presented in the next few columns.

There were some issues raise during our evaluation process. One of the Website,

Websites	Checks		Manual detection	Auto detection	Auto Complete Off "tell" Sign	Up and Down Keys "tell" Sign	Updates "tell" Sign	Clear List "tell" Sign	Remote Scripting "tell" Sign
	Manual detection	Auto detection							
yahoo.com	1	1	1	1	1	1	1	1	0
google.com	1	1	1	1	1	1	1	1	1
youtube.com	1	1	1	1	1	1	0	1	1
live.com	1	1	1	1	1	1	0	1	1
msn.com	0	0	0	0	0	0	0	0	0
myspace.com	0	1	1	1	1	1	1	1	1
wikipedia.org	0	0	0	0	0	0	0	0	0
facebook.com	0	1	1	1	1	1	1	1	1
blogger.com	0	0	1	0	1	1	1	1	1
orkut.com	0	0	0	0	0	0	0	0	0
rapidshare.com	0	0	0	0	0	0	0	0	0
baidu.com	1	1	1	1	1	1	1	1	1
microsoft.com	0	0	0	1	1	1	1	1	1
qq.com	1	X	1	0	1	1	1	1	1
ebay.com	1	1	1	1	1	1	1	1	1
hi5.com	0	0	0	1	1	1	1	1	1
aol.com	0	1	1	1	1	1	1	1	1
mail.ru	1	1	1	1	1	1	0	1	1
sina.com.cn	1	1	1	1	1	1	1	1	1
fc2.com	0	0	0	0	1	1	1	1	1
<b>Total</b>	<b>9</b>	<b>11</b>	<b>13</b>	<b>13</b>	<b>16</b>	<b>13</b>	<b>15</b>	<b>15</b>	<b>15</b>

**Table 1:** Evaluation results for the ASL widget detection.

qq.com, uses a licensed third party widget that made it impossible to retrieve the code. As a result, we have decided to exclude this Website from our analysis. ASL widgets were incorrectly detected on three out of nineteen Websites. Two out of these three Websites were myspace.com and facebook.com. These Websites reuse the same external JavaScript files over multiple Web pages. In these external JavaScript files, many functions/classes are included even if they were not used by the Web page, which causes false detection to occur. After conducting a thorough investigation on both of these Web pages, it was found that the features of the ASL widget do exist within the external files, but they were never called or used by the

Web pages. This is an issue concerning the style of organisation that may vary from one Website to another. Another reason for this issue is the weakly-typed client-side scripting languages used over the Web. The third Website, `ao1.com` had a lot of Web widgets implemented on it. Thus small bits of code from the different Web widgets, combined caused our approach to give a false positive results for this Website.

The results presented in table 1 gives us a positive sign that our approach is feasible. The Web is a heterogeneous set of technologies, guidelines, and recommendations that undergoes constant and combinatorial changes. Our method correctly detected all the ASL widgets that were present, but the results highlighted a 16% false positive detection rate that requires further investigations to fine tune our methods. We are convinced that this set of results has proven that our approach is feasible, however further investigations should be pursued to refine our methodologies.

## 5 Carousel Widget

The Carousel widget is a presentation model to display content on a Web page, so that only one or a few items in a list is displayed at a time. This widget provides the users with the control to scroll through the list of content, focusing on the information he/she wants to concentrate on.

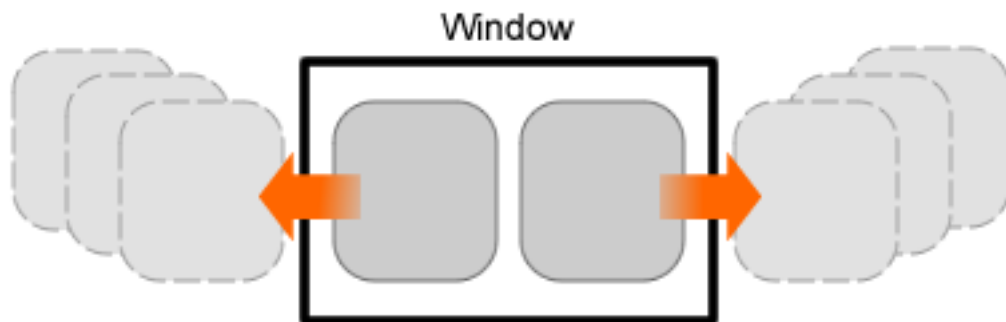
Three basic features are provided on the Carousel widget. Items in the list of content can be rotated round in both directions by controlling a set of controls on the Carousel widget. This is done by either the ‘next’ and ‘previous’ buttons, or the ‘up’ and ‘down’ buttons. Both sets of controls provide the user with the power to control the direction the content should rotate, depending on how the widget was developed. Thirdly, whenever no directional instruction is given by the users, the Carousel widget will remain at the last requested item. In the rest of the report, the term ‘Next’ button will be used to refer to the ‘next’ or ‘down’ button, and the term ‘Previous’ button will refer to the ‘previous’ or ‘up’ button. An example of a Carousel widget is presented in the non-shaded region in figure 6. This example shows a small version of the Carousel widget with the ‘next’ and ‘previous’ buttons to the left of the widget’s display window.

Figure 7 gives a visual illustration of the process and parts of a Carousel widget. The area where content is presented by the widget will be described as the ‘window’. The content in the ‘window’ will be updated whenever a new item within the list of content is requested by the user. This is the area on the Web page where the Carousel widget will affect micro-content to update dynamically.

The Carousel widget is chosen for the evaluation due to the nature of its behavior: the content in the window will update dynamically whenever the ‘next’ or ‘previous’ button is clicked by the user. As the features of the Carousel widget are basic, it has a number of similar properties with other Web widgets such as the Slide Show and Module Tabs. So if the definition of the Carousel widget is not dealt with carefully, this can cause ambiguity in our methodology, and a high false positive detection rate can be expected. Taking the Slide Show and the Module Tabs widgets instance, these widgets have very similar, sometimes identical properties with the Carousel widgets,



**Figure 6:** An example of the Carousel widget (non-shaded region) in action on blogger.com. The arrows are used to move to the previous or next item.



**Figure 7:** Visual description of the operations of a Carousel widget. The Window is where the items are displayed to the user.

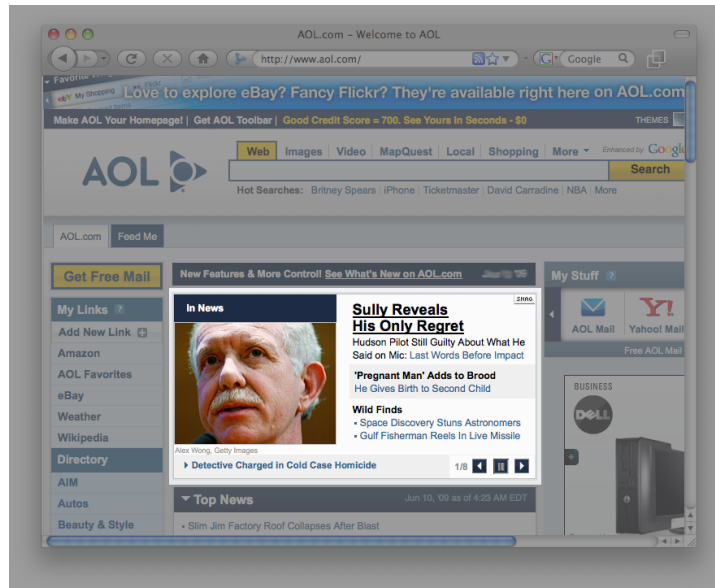
and this cause false positive detection in our methods. In the next two sections, we will examine these widgets in more depth.

### 5.1 Slide Show Widget

A Slide Show widget is another model for presenting a list to users. This widget is a superset of the Carousel widget. Unlike the Carousel widget, it provides the users with automated features, and more control over the flow of content. An example of a Slide Show widget is shown in the non-shaded region in figure 8. In this example the controls are found at the bottom right hand corner of the widget's user interface.

Commonly after loading the Web page, the Slide Show widget will play the

content automatically. The user can pause or resume playing it at any point. Some Slide Show widgets even has a fast forward/skip button to allow the user to progress through the content at a faster pace. Similarly to the Carousel widget, the user can return to the previous content or progress to the next content in the list whenever he/she wants to. Besides the controls for controlling the display of content, the infrastructure of Carousel widget is identical to the Slide Show widget. This can lead to false detection of both of these types of widgets if not dealt with properly.



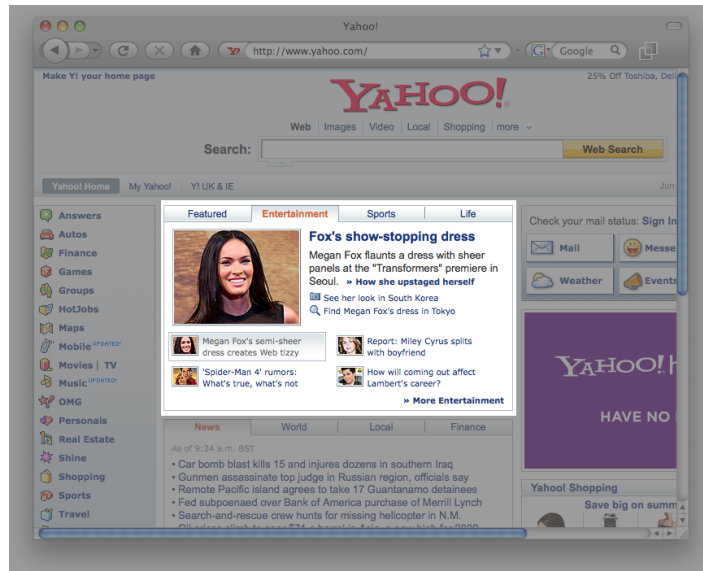
**Figure 8:** An example of the Slide Show widget (non-shaded region) in action on aol.com

In order avoid confusion, we have defined a Carousel widget to be a Web widget with a window, a ‘next’ button, and a ‘previous’ button. However these are basic features that are common to many Web widgets as mentioned. The main difference between the Slide show widget and the Carousel widget is a Slide Show widget has additional features that a Carousel widget does not have. These are the ‘Pause’ and ‘Play’ features that allow users to inform the widget whether they wish to navigate through the content manually, or let the widget to do it automatically.

## 5.2 Module Tabs Widget

The Module Tabs widget is another widget that has similar properties with the Carousel widget. Unlike the Carousel widget that uses a ‘Next’ or ‘Previous’ button to navigate through the list of content, the Module Tabs widget allows users to select directly the information they want to read. This means that users can skip the irrelevant content, and navigate directly to the information they wish to read. Normally this widget is used to convey much more information than the Carousel widget. An example of a Module Tabs widget is shown in the non-shaded region

of figure 9. In the given example, the users can change the displayed content by clicking on the relevant tabs at the top of the widget.



**Figure 9:** An example of the Module Tabs widget (non-shaded region) in action on yahoo.com

The main difference between the Carousel and the Module Tabs widget is the way it allows the user to navigate through the content. However, both widgets use a similar method to present the information to the user. This is done through the means of changing the content in the widget's window.

### 5.3 The “tell” Signs

“tell” signs are used to identify the Carousel widget on a Web page. They were defined through extensive investigation of Web pages containing Carousel widgets, Yahoo! Developer Network's design pattern library<sup>7</sup>, and Welie's pattern library<sup>8</sup>. Additional refinements were also introduced to the “tell” signs, to provide a better distinction for the Carousel widget.

Four “tell” signs are used to harvest the results for this experiment. A description of the purpose, methods, and the flow of processes for each “tell” sign is presented.

#### Increment and Decrement “tell” Signs (ts1 and ts2)

The Increment and Decrement “tell” signs search for clues that the ‘Next’ and ‘Previous’ button functions exist within the JavaScript code. The ‘Next’ button can be thought of as a function that increments the value of a pointer that is pointing to a location within an array where the list of contents is stored. To find this “tell”

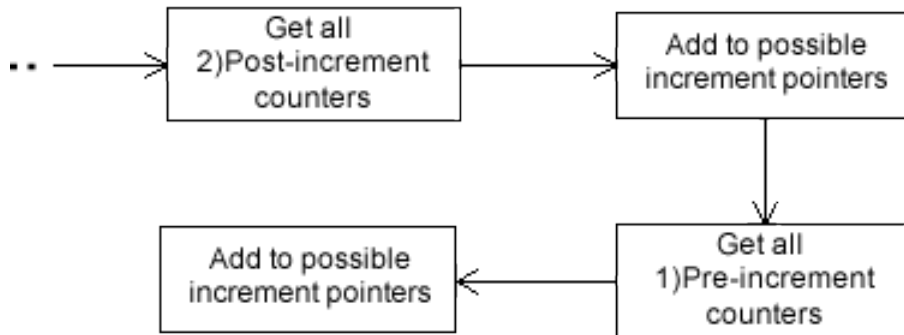
<sup>7</sup><http://developer.yahoo.com/ypatterns/>

<sup>8</sup><http://www.welie.com/patterns/>

sign, an attempt is made to detect a pattern where a variable is incremented in the JavaScript code. Two methods can be employed by the developers to code an incremental pointer; 1) pre-increment, and 2) post-increment. To search for these clues, the following two regular expressions were used respectively.

```
/(\\s|;)\\+\\+([-_0-9a-z]+)/i      /* pre-increment */
/(\\s|;)([-_0-9a-z]+)\\+\\+/i      /* post-increment */
```

The process used to determine whether variables values are incremented is shown in figure 10. The ‘Get all 1) Pre-increment counters’ block and ‘Get all 2) Post-increment counters’ block in this figure are where the pre-increment and post-increment regular expressions were applied respectively. Each occurrence of the identified post-increment counter and pre-increment counter variables is concatenated with the overall list of possible increment variables array.



**Figure 10:** Carousel widget increment “tell” sign (ts1) detection flow

The same process is used to search for a decremental pointer within the Web page’s source code. However, this time the following two regular expressions are used instead to search for the pre-decrement and post-decrement pointers respectively. Similarly, each occurrence of the identified post-decrement and pre-decrement counter variables will be concatenated with the overall list of possible decrement variables array.

```
/(\\s|;)(\\-\\-)([-_0-9a-z]+)/i      /* pre-decrement */
/(\\s|;)([-_0-9a-z]+)\\-\\-/i      /* post-decrement */
```

### True Pointers “tell” Sign (ts3)

To ensure that the variables identified in *ts1* and *ts2* are true pointers that point to a location within a list/array, the True Pointers “tell” sign is used. This “tell” sign will ensure that the suspected variable is used in the code to refer to a location within the list/array. Before attempting to process this “tell” sign, our method will

filter the candidate variables identified by *ts1* and *ts2* (see figure 11), so that only variables identified by both *ts1* and *ts2* will proceed to *ts3*.

Two methods are used to search for the True Pointers “tell” sign. The first method searches for clues that conditional expressions were enforced on the suspected variables. Commonly, conditional expressions are used by developers to ensure that the pointer’s value is restrained to the locations within the list/array. Thus, the following regular expression is employed to search for these clues. In this regular expression, the concatenated array ‘`$possible[$i]`’ stores the names of the variables that were identified by the first two “tell” signs. Using this regular expression, we can differentiate the variables that were used by the developers to enforce certain conditions in their processes.

```
/if\([\w\W]*\s*(==|<=|>=|>|<)?".$possible[$i].
\s*(==|<=|>=|>|<)?[\w\W]*\)/i
```

The second method checks if the suspected pointers are used to refer to an address location within the list/array. This is done using the following regular expression. In this regular expression, the variable ‘`$ptr[$i]`’ is the list of variables suspected to be the pointer after it was filtered by the first method in this “tell” sign.

```
/([_0-9a-z]+)\[".$ptr[$i]."\]/i
```

#### Show and Hide “tell” Signs (ts4)

The Show and Hide “tell” sign searches within the Web page’s source code for attributes that affect the displayed content in the widget’s window. This can be done using a few methods, and checks are required to ascertain how the selected content will be displayed on the Web page. One of the commonly employed methods to control the content displayed in the widget’s window is changing the CSS attributes for a particular element.

```
/".$fdlist[$ltc]."\[".$fptr[$ptc]."\]\.style\.display\s*
=\s*(\'|\\")block(\'|\\")/i
```

```
/".$fdlist[$ltc]."\[".$fptr[$ptc]."\](\s)*(\|)?\.style\.
display\s*=\s*(\'|\\")none(\'|\\")/i
```

The first regular expression above searches for signs that the ‘display’ attribute in the CSS code is set to ‘block’. The ‘display’ attribute in CSS specifies how the appearance of the DOM element should be generated. It was observed that for a Carousel widget, commonly the content element in the display Window is generated as a block box. The second regular expression above searches for clues that an attempt was made to hide the specified element within the DOM. Either of these regular expressions must be satisfied for this “tell” sign to be true.

## 5.4 Assumptions and Rules

A number of assumptions were made when designing the “tell” signs for detecting the Carousel widget in this experiment. One of the main issues is determining the granularity of the design patterns. Due to the heterogeneous combinations of styles, formats, technologies, recommendations, and guidelines that make up the Web, defining the granularity of the patterns must be fine enough so that it will be flexible and reusable. Therefore, the agility of our approach depends a lot on our defined axiom definition for the different objects. Consider, for example, the button object. Unlike conventional programming, over the Web this can be in the form of either a form button, a hyperlink with plain text, or a hyperlink with images. Furthermore a hyperlink with images can be a link to another Web document, or a link that interacts with the client-side scripts, or a link that redirects the user to another part of the same Web document. This adds complexity to the concepts and relationships between the objects. In this case, it will be assumed that the button object is an element within a Web page, that calls a portion of the JavaScript code when activated by a user event.

Our method covers only Carousel widget developed in JavaScript. This can be expanded, if required, for other scripting languages using the same concept. Equation 2 introduces a rule to govern our approach for detecting the Carousel widget.

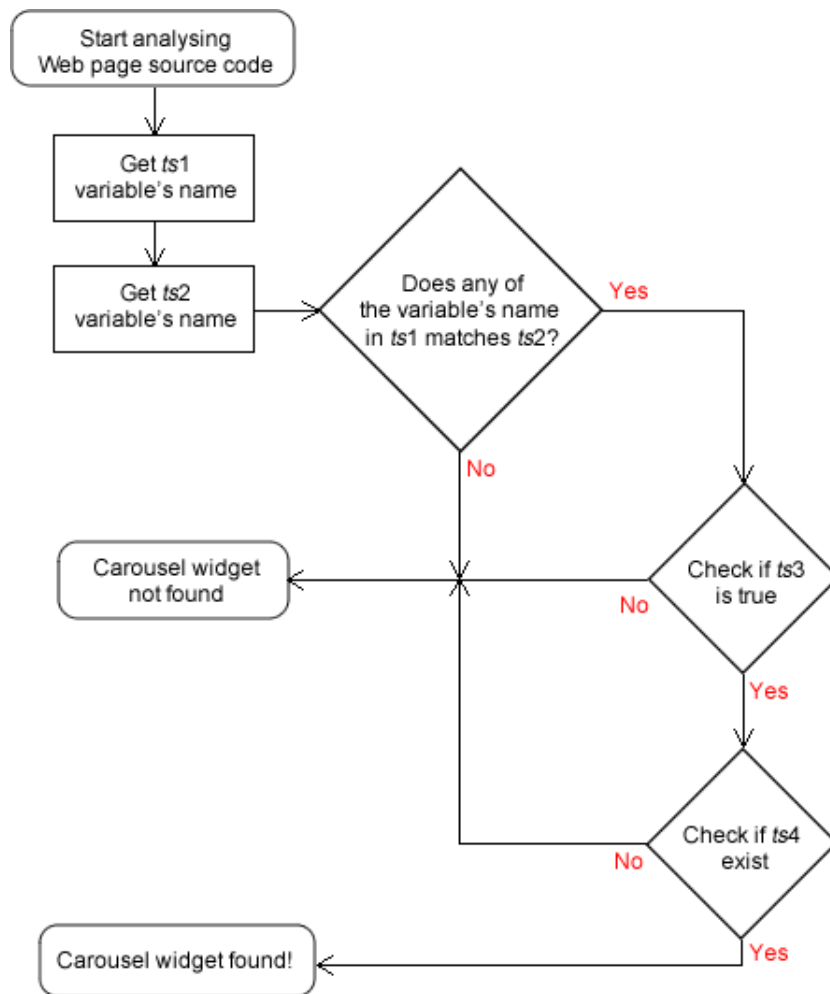
$$ts1 \wedge ts2 \wedge ts3 \wedge ts4, \quad (2)$$

where  $ts1$  is the Increment “tell” sign,  $ts2$  is the Decrement “tell” sign,  $ts3$  is the True Pointer “tell” sign, and  $ts4$  is the Show and Hide “tell” sign. The rule says that all the four “tell” signs must be true before it can be assumed that the Carousel widget exists on the Web page.

The identification process for the Carousel widget begins with searching for both  $ts1$  and  $ts2$  as illustrated in figure 11. If at least one of the detected variables in  $ts1$  and  $ts2$  matches, it will proceed to  $ts3$  for separating the true pointers from the random counters. If none of the detected variables in  $ts1$  and  $ts2$  match, then it will be assumed that there is no Carousel widget on the Web page. Next if at least one true pointer is pick up in  $ts3$ , then  $ts4$  will be conducted, otherwise again no Carousel widget will be assumed to be on the Web page. Finally, if the Web page passes all of the first three “tell” signs, then  $ts4$  will ensure that the Carousel widget’s displaying feature is present on the Web page before it assumes that the Carousel widget existed. It is worth noting that the processing sequence for  $ts3$  and  $ts4$  will be repeated for each candidate variables, that is identified by both  $ts1$  and  $ts2$ .

## 5.5 Limitations

This experiment has been scoped to analyse code formatted in Hypertext, CSS, and JavaScript only. Thus Carousel widgets that are developed in Flash, Shockwave, VBScripts, or in any other format are not covered.



**Figure 11:** Carousel widget detection’s “tell” sign process flow chart

The method employed to detect the Carousel widget is partly derived from extensive studies on the commonalities between the Websites with Carousel widget from our experimental data set. Therefore, our approach is limited to these methods determined.

## 5.6 Results and Discussions

The results presented in table 2 detail both the automated and the manual detection results for comparison, and the results for the individual “tell” signs.

The first column of table 2 list, in order, the top twenty Websites selected for our experimental data set. The second column presents the results of the manual detection for the Carousel widget. This is done manually by a human, viewing the individual Websites to check if the Carousel widget existed. The ‘Auto detection’ column lists the final conclusion drawn using our methodology, as to whether the Carousel widget exists on a Web page. In the next few columns, the results for the

Websites \ Checks	Checks		Increment “tell” sign	Decrement “tell” sign	True pointers “tell” sign	Show and Hide “tell” sign
	Manual detection	Auto detection				
yahoo.com	0	0	1	1	0	0
google.com	0	0	1	0	0	0
youtube.com	0	1	1	1	1	1
live.com	0	0	1	0	0	0
msn.com	0	X	1	1	X	X
myspace.com	0	0	1	1	0	0
wikipedia.org	0	0	0	0	0	0
facebook.com	0	X	1	1	X	X
blogger.com	1	1	1	1	1	1
orkut.com	0	0	0	0	0	0
rapidshare.com	0	0	1	0	0	0
baidu.com	0	0	1	0	0	0
microsoft.com	0	1	1	1	1	1
qq.com	0	0	1	1	1	0
ebay.com	0	0	1	1	1	0
hi5.com	0	X	1	1	X	X
aol.com	0	X	1	1	X	X
mail.ru	1	1	1	1	1	1
sina.com.cn	0	0	1	1	0	0
fc2.com	0	X	1	1	X	X
<b>Total</b>	<b>2</b>	<b>4</b>	<b>18</b>	<b>14</b>	<b>6</b>	<b>4</b>

**Table 2:** Evaluation results for the Carousel widget detection

four “tell” signs for each Website is presented.

To interpret the results in table 2, a ‘1’ symbolised a true, a ‘0’ symbolised a false, and a ‘X’ symbolised that the investigation for some reason was ignored. For an example, lets look at **blogger.com**. since all the columns for this Website is ‘1’, then Manual detection and all the four “tell” signs are true too, so auto detection is true as well.

The Carousel widget is a piece of code that uses a combinations of different languages to work. Hence, developers can employ a vast range of styles and practices to shape this widget. This makes it difficult to detect for all the different styles, and practices of the Carousel widget. A close examination of table 2 highlights that the results for the Auto and Manual Detection do not tally, but a 100% detection rate is

achieved when the Carousel widget is present. Websites such as `youtube.com` and `microsoft.com` were falsely detected. After investigating the types of Web widgets used by both Websites, it was noticed that both Websites contain the Module Tab widget. As explained in section 5.2, some parts of the Module Tab model is similar to the Carousel model, and this issue has contributed partly to the false positive detection. On `youtube.com`, the combination of clues from both the ASL and the Module Tabs widget led to the false detection of the Carousel widget.

It is worth noting that the Carousel widget was also used on `ebay.com`; however the developers of `ebay.com` chose to build them with Flash which is not included in this study. Issues with the limited memory allocated for each Web page by the PHP engine were also highlighted. This meant five Websites with a very large amount of code that uses a lot of variables, could not be evaluated. This is because of the amount of memory allocated by the PHP engine when interpreting a Web page. Thus for these Web pages, only the first two “tell” signs were evaluated, and then the system froze when it ran out of memory. Since only the first two “tell” signs were investigated, the results for these five Websites will not be taken into consideration in our evaluation.

The five Websites that were ignored are Websites with at least one column in table 2 that is marked with ‘X’. These are `msn.com`, `facebook.com`, `hi5.com`, `aol.com`, and `fc2.com`. No further work is followed up for these Websites since the majority of the Websites are fine to evaluate this investigation.

Since only 25% of the Websites from our experimental data set are affected by the memory issue, our conclusion was made with the remaining 75%. The results displayed in table 2 gave a promising sign for our detection method. We managed to successfully detected all the Carousel widgets that were present from the remaining 15 Websites, but two false positive were detected. Comparing these results with the ASL widget in section 4.4, the Carousel widget gave a more positive results.

### Issues with PHP Memory Limit

The memory allocated to each Web page by the PHP engine imposed a limitation to our methodology. The PHP engine allocates each Web page a maximum of 8MB of memory when interpreting them. This becomes an obstacle when the system tries to comprehend large amounts of code that uses a lot of variables to complete its task.

The memory issue occurred because our method uses many arrays to store the suspected pointers and variable names that could be the list of content; the list of content on its own is an array already. Therefore, multi-dimensional arrays are expected to be employed for our methodology. Memory management is another issue with PHP, because this is managed by the interpreter. An attempt was made to force the PHP memory limit to the maximum, but this did not improve the performance much, as it soon ran out of memory again. Since the intent of the experiment is to provide a proof of our concept, and this problem only affected a quarter of our experimental data set, the memory issues faced with PHP were not pursued.

## 6 Future Work

The results presented in tables 1 and 2 give a promising sign that our approach is suitable, but more work will be required to refine our approach. A common problem is spotted with both set of experiments: little bits of codes or clues from different areas within the code can be picked up as our “tell” signs. This is because at the moment there is no way to determine if the clues are related to the objects on the widget’s interface.

### 6.1 Time Out “tell” Sign

The four “tell” signs used are, at times, not sufficient to uniquely distinguish the Carousel widget from other Web widgets with similar properties. After investigating the problem, it was found that the Carousel widget can be easily confused with a Slide Show widget. This is because the Carousel widget’s properties are the subset properties of the Slide Show widget. Thus more “tell” signs are required for further distinguishability. It is realised that for a slide show widget, the content displayed can be controlled either automatically or manually. For automation, in this case it will require some form of timer which can be done by using the timeout function. To detect for the Time out “tell” sign, the following regular expression may be employed.

```
/setTimeout\([\\"|\\"'+[\W\w]+[\\"|\\"']+\, \s*[0-9]+\)/i
```

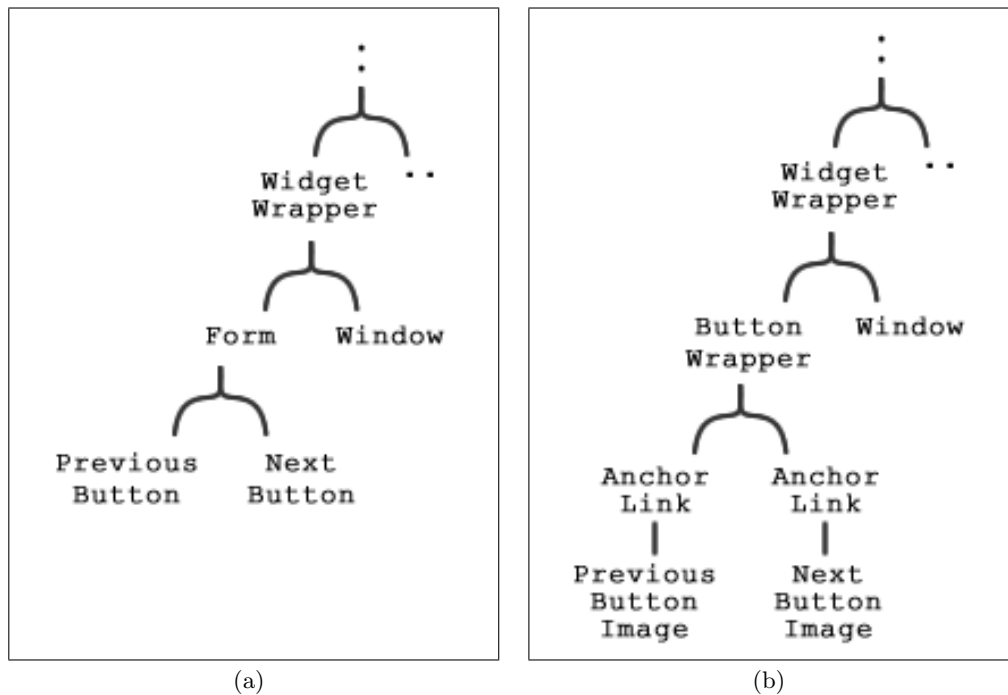
This regular expression will search the JavaScript code for the possibility that the ‘setTimeout’ function is used. It also checks within the function for two parameters: The first one is the instructions for what happens after the time out, and the second parameter specifies in milliseconds how long will the script lasts before it times out.

### 6.2 Static Object Model “tell” Sign

The Static Object Model “tell” sign will help the system to identify coherent graphical user interface (GUI) objects of the Web widget from the Web page’s source code. By mapping the static DOM elements, it will enable the system to interlink the mapped elements with the related JavaScript and CSS code.

To overcome issues with identifying the different patterns that form the widget’s GUI, it is suggested that a static graph should be plotted from the document’s object model. This will allow us to group related elements through hierarchical classification. However, the mutation to the document object model will not be analysed, and no modification to the static graph will be made during the course of the Web page as such work has been covered by Brown and Jay [4].

For the Static Object Model “tell” sign, understanding the semantics of the elements that forms the GUI for the widget, will provide the means to understand the relationship between the client-side scripts, and the GUI elements. By doing this it will streamline the area within the code where we should examined. Thus removing the non-related part of the original code, and reducing the possibility for a false positive detection.



**Figure 12:** Static graph examples for the Carousel widget. Subfigure (a) shows an example of how the static graph will look like when form buttons were used for the ‘Next’ and ‘Previous’ buttons. Subfigure (b) shows an example of how the static graph will look like when images were used for the ‘Next’ and ‘Previous’ buttons.

This method will allow the different forms of a button object (form buttons, and hyperlinks in form of text or images) to be grouped together with the Carousel widget’s window object. By doing this, it will be possible for us to relate the button object with a part of code that forms the widget’s engine, and the element that is meant for displaying the content (the widget’s window). The examples presented in figure 12 demonstrate that by plotting the static graph of the first delivered Web page, the widget’s interface elements can be easily grouped together by the means of a hierarchical structure.

In some cases, the ‘Window’ node can be replaced with the actual list of content, where each item in the content list will be a node in the graph under the Widget Wrapper’s node. Then, using CSS the developer can control whether the node should be presented or hid from the users. Through the means of a hierarchical structure an assumption that some form of ‘wrapper’ will be used to interconnect the controls of the widget with the window of the Carousel widget. Hence, by identifying the wrapper, we can assumed that the elements after/under the wrapper element have an association with it.

This work can be further divided into two parts, so that we can comprehend the declarative code structure (hypertext) together with the procedure code structure (we refer to JavaScript for this study). By using this method, it will improve the understanding of the process and elements coherency.

### 6.3 Annotating the Source Code

Annotating the source code adds derived documentations from chunks of code before our method comprehends it. This concept was borrowed from Takagi et al. in [13], where they demonstrated the power of annotation to develop an accessibility transcoding system. The process of annotating the source code is suggested as a pre-processing stage, to arrange, and add semantical information derived from small portions of the code, so that it will provide useful machine interpretable documentations for our method to improve code comprehension. This method will assist the code comprehension process, and we will only need to comprehend the code in great depth once, and not every time when we need to refer to it.

On the contrary, this method can be complex and it will take up a lot of additional computational power. Thus, for the purpose of this study, this method may not be the most favorable.

### 6.4 More Comprehensive Evaluation

Finally, we proposed a more comprehensive evaluation to be conducted, so that more thorough investigation can be concluded. So far evaluations have been performed on a selection of top twenty popular Websites. Although the evaluation methods applied for this study has given a good representation of how our methodologies will perform in the general Web [6], to give a more convincing evidence that our approach is feasible, we need to run the evaluation on a larger scale.

## 7 Conclusion

Web 2.0 has given the users of the Web rich internet applications, that source data and remix them from multiple sources, so that content on a Web page can be updated in small chunks. These concepts introduce changes to the conventional ‘Page Model’ that most of us are used to [10], and indirectly makes the Web less accessible. Displayed content on a Web page that use the Web 2.0 concepts, update their presented content in the micro content format rather than reloading the entire Web page. Hence, assistive technologies must adapt to these changes so that they will be able to provide accurate information to people with disabilities.

In this study, we demonstrated the practicability of including a remedial process at the developer end, so that areas where micro-content may be updated can be identified, and Web widgets that produce inaccessible content can be modified into an accessible form. Nevertheless when designing these experiments, issues with defining the granularity of the design pattern, and traceability of the design patterns from the Web page’s source code were highlighted.

From the first experiment that detects the usage of the ASL widget, the evaluation results showed promising signs that our approach is applicable. The evaluation shows that the detection for the ASL widget has successful detection all the ASL widget present, but a few false positive detection were also found. However, further investigation is required to understand how to improve the issues with the different languages that existed on different layers that made the widget work, and how the

different document formats relate to one another to form the Web page. Suggestion such as introducing a Static Object Model “tell” sign in section 6 may provide the solution to improve the 16% false positive detection.

The second experiment that attempts to detect the Carousel widget has brought up critical issues with the scripting language used for the evaluation. It highlighted that PHP allocates only 8MB to each Web page during run time. This issue limited our evaluation capabilities, and 25% of the Websites from our experimental data set were unable to complete the evaluation. Thus, this experiment was based on the remaining 75% of the Websites. In this experiment, all the Carousel widgets present were detected, however, two false positive were also present. These results shone a positive light on our method employed, and when compared with the ASL widget detection methods, a lower false positive detection was noticed.

Our methods eliminate false negative detections, but to deal with false positive detections, further investigations should be followed. Such studies will reduce the ambiguity in our definition for a widget, e.g., the Carousel widget, and provide a more robust methodology. A well defined definition of the widget is an essential element to ensure a high successful detection rate. Both Web widgets experiments, successfully detected all the ASL and Carousel widgets, and through these evaluations, issues were highlighted and the suggested improvements for better detections of the two widgets are presented in the Future Work section. It is strongly suggested that this work should be pursued to improve the reliability of our method, and to evaluate them on a larger scale, so that the diversity of our method can be tested, and their weaknesses exposed.

## References

- [1] C. Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.
- [2] Apple Developer Connection. Remote Scripting with iFrame. <http://developer.apple.com/internet/webcontent/iframe.html>, 2009.
- [3] J. Bosch. Design patterns as language constructs. *Journal of Object-Oriented Programming*, 11(2), 1998.
- [4] A. Brown and C. Jay. SASWAT Technical Requirements. <http://hcw-eprints.cs.man.ac.uk/79/>, The University of Manchester, October 2008.
- [5] S. Canter. Understanding Client-Side Scripting. <http://www.pcmag.com/article2/0,2817,1564972,00.asp>, April 2004.
- [6] A. Q. Chen. Web evolution: Method and materials. <http://hcw-eprints.cs.man.ac.uk/74/>, The University of Manchester, September 2008.
- [7] A. D. Edwards. *Web Accessibility: A Foundation for Research*, chapter Assistive Technologies, pages 141–162. Springer, 2008.
- [8] D. Flanagan. *JavaScript: The Definitive Guide*. O’Reilly, 4th edition, 2002.

- [9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [10] D. Maurer. Usability for rich internet applications. *Digital Web Magazine*, February 2006.
- [11] G. Murray. Asynchronous javascript technology and xml (ajax) with the java platform. <http://java.sun.com/developer/technicalArticles/J2EE/AJAX/>, October 2006.
- [12] T. O'Reilly. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. *Communications & Strategies*, (65), 2007.
- [13] H. Takagi, C. Asakawa, K. Fukuda, and J. Maeda. Site-wide annotation: reconstructing existing pages to be accessible. In *Assets '02: Proceedings of the fifth international ACM conference on Assistive technologies*, pages 81–88, New York, NY, USA, 2002. ACM.