Towards Web Application Mobilization via Efficient Web Control Extraction

Shuai Wang, Wensheng Dou, Guoquan Wu, Jie Wang, Chushu Gao, Jun Wei, Tao Huang
Technology Center of Software Engineering
Institute of Software at Chinese Academy of Sciences, Beijing, China
{wangshuai, wsdou, gqwu, wangjie12, gaochushu, wj, tao}@otcaix.iscas.ac.cn

ABSTRACT

Traditional web applications are not suitable for mobile devices, because mobile devices are usually equipped with small screens and use slow and expensive mobile network. Existing approaches to adapt web applications to mobile devices include reconstructing particular web applications, which requires much additional work, or adapting only views of web pages, which still requires a lot of network bandwidth. In this paper we propose an approach which extracts a part of a web page as an executable web control efficiently. Our approach only monitors the execution of user code, builds a dependency graph of executed user code, and performs slicing upon the dependency graph. The evaluation on two real-world web applications shows that our approach is able to extract executable web controls efficiently, and visiting extracted web controls instead of the whole web page saves 99% and 23% of bandwidth separately.

Categories and Subject Descriptors

D.2.7 **[Software Engineering]**: Distribution, Maintenance, and Enhancement – *Restructuring, reverse engineering, and reengineering*

General Terms

Algorithms, Theory.

Keywords

Web Application, Web Control, Code Extraction, Program Slicing

1. INTRODUCTION

More and more people access web information through mobile devices, but traditional web pages are developed for desktop devices, such as PCs, which are not suitable for displaying in mobile devices directly. Traditional web pages always contain many kinds of information, such as news, navigations, ads, and so on, while mobile devices aim to display single kind of information [1], because of their small screens, limited network and power.

- Most mobile devices use 3–3.5 inch screens [2], while mainstream desktop devices use screens around 20 inches. Therefore, web pages designed for 20 inch screens will be zoomed out to fit small screens of mobile devices, which degrades the readability of original web pages.
- Mobile devices always use wireless network, which is bandwidth-limited, unstable, and charged by consumption. Traditional web pages are designed for desktop devices using unlimited network, and they always contain information as much as possible, which causes a huge size for a single web page. For example, users may download 19.59 MB resources before visiting the home page of SinaNews [3]. Therefore, visiting traditional web pages via mobile devices is a time-consuming and money-consuming work.
- Mobile devices uses batteries as their power supplier, which is one of the short boards of mobile device usability. Some smartphones last less than 2 hours only if they continuously download files over 3G connections [4]. As mentioned above, traditional web pages always have huge sizes, which causes huge cost of power on downloading these web pages.

A lot of work has been done to solve these problems. Some organizations build an individual mobile version of the original web site, which is totally different, such as sina wap [5]. While building a mobile version does solve the problems completely, it requires additional budget. Some organizations present develop tools to reduce the budget. SiteApp [6] provided by Baidu [7] helps developers to build mobile version of an existing web site in only four simple steps, but it only works on certain type of web sites. IBM [8] presents Worklight [9], which is claimed to support all kinds of web sites, but it requires a lot of developing efforts. Some research works have been presented to solve these problems too [1] [10] [11] [12], which supply readable views of web pages via displaying only part of web pages fitting the screen at one time. These works mainly aim to solve the problem caused by differences of screen sizes, and they don’t care about the high cost of bandwidth and power caused by visiting web pages of huge size.

It is a better way to download only user-interested part instead of the whole one while visiting web pages via mobile devices, as part of web pages fits better to the mobile screens than the whole ones, and downloading only a part saves bandwidth and power obviously. To achieve this goal, we present an approach to perform efficient web control extraction for web application mobilization. Our approach is inspired by the following observation: web applications are event-driven [13]. Event handlers implemented in JavaScript are used to interact with users, which have to be registered to some HTML DOM elements firstly, and then will be triggered by events fired on it or its child elements. Based on this observation, our approach extracts selected DOM elements as presentation of specified web control, and infers event handlers of specified web control by monitoring execution of event handlers; in order to perform the process efficiently, we monitor only user defined code. Compared to existing extracting technique [16] which monitors all the executed code containing the JavaScript code inside the third-party library, the proposed approach can extract web controls from web applications of a large scale efficiently.

In summary, our contributions are as follows:

- We propose an approach to efficiently extract web controls from web pages.
- We implement a prototype system for our approach. Our prototype system allows developers to extract web controls from real-world web pages.
- We perform two case studies on popular real-world web applications, and the result shows that our approach is able to extract executable user specified web controls efficiently.

The rest of the paper is organized as follows. We illustrate the motivation of our approach by a real-world example in Section 2, and the overview in Section 3. In Section 4, we describe our approach in detail. We show how we have evaluated the approach Section 5, and discuss the limitations of the approach in Section 6. Related works are presented in Section 7. Finally we give a conclusion in Section 8.

2. Motivation Example

In this section, we introduce a real-world example that will be used throughout the paper. The example is a simplified version of part of Yahoo home page [14], and is built on jQuery [15].

Yahoo is one of the most popular web sites of the world, it is globally known for its web portal and search engine. We build a simple web page following Yahoo home page, which consists of a search engine entry and a hot news viewer, as shown in Fig. 1. The search engine entry lies at the top of page, which consists of a text input control and a submit button control. The hot news viewer lies at the bottom, which consists of a large picture viewer marked with news title and a hot news list containing five small pictures. Users are allowed to submit a search request by typing search keywords and then clicking the submit button, or switch the large picture by clicking on some small picture in news list, or navigate to news detail page by clicking on the large picture. Additionally, the page displays hints to direct user interaction, e.g., when a user move his mouse over the text input control, a hint displaying 'Enter your search term' appears.

Fig. 2 shows the HTML code of the example. There are two child DIV elements of *body*, i.e. *searchWrapper*, and *main_story*. The *searchWrapper* DIV (line 5 - 10) contains elements of the search engine entry. The *main_story* DIV (line 11 - 41) contains elements of the hot news viewer, and it is composed of two sub DIV elements, *main_story_content* (line 12 - 23), which is served as the large picture viewer, and *footer-section* (line 24 - 40), which is served as the hot news list.

Fig. 3 shows the JavaScript code of the example. It defines four functions. The first one, *updateMainStory* (line 1 - 3), updates the content of the large picture viewer by the received msg. *updateMainStory* is asynchronously invoked by the second one, *showMainStory* (line 4 - 8), which asynchronously updates content of the large picture viewer according to *data* attribute of the DOM element that triggers the function. The third one, *showHint* (line 9 - 15), displays the hint attribute as the hint information to users, and the last one, *hideHint* (line 16 - 18), hides the hint information. The last four lines of code (line 19 - 22) register the above four functions to some DOM elements as their event handlers. Line 19 registers *showMainStory* as the *click* event handler of each item of the hot news list (*.footer-img-holder*). Line 20 - 22 add features of show and hide hint to each

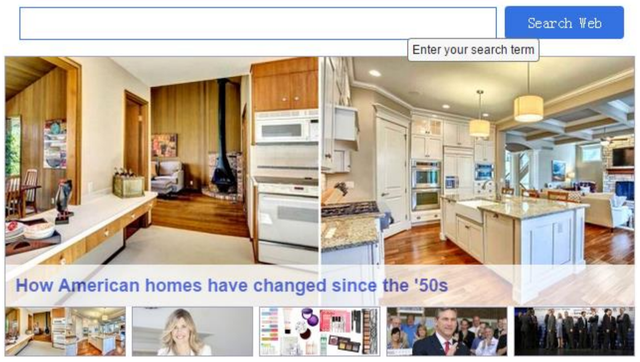
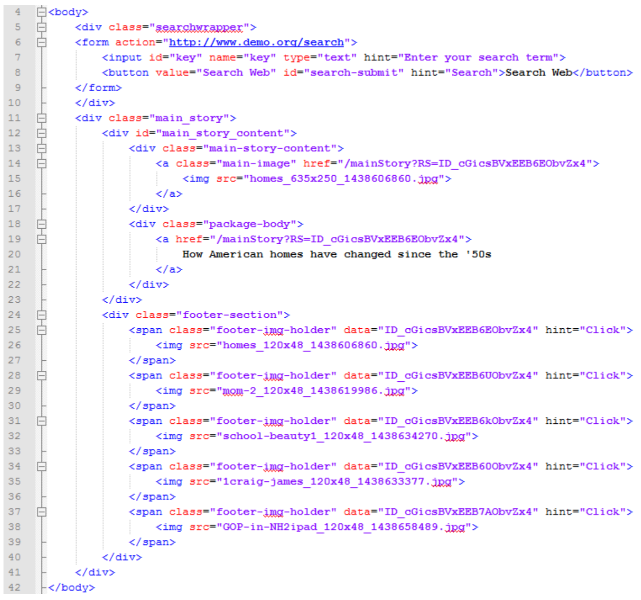
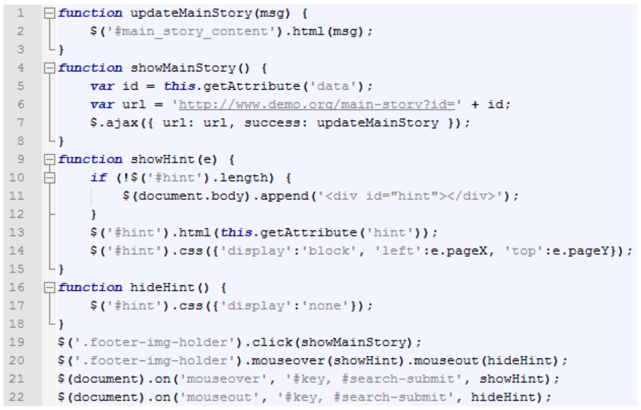


Figure 1. The Web Page of the Example



```
4   <body>
5       <div class="searchwrapper">
6       <form action="http://www.demo.org/search">
7           <input id="key" name="key" type="text" hint="Enter your search term">
8           <button value="Search Web" id="search-submit" hint="Search">Search Web</button>
9       </form>
10      </div>
11      <div class="main_story">
12          <div id="main_story_content">
13              <div class="main-story-content">
14                  <a class="main-image" href="/mainStory?RS=ID_cGicsBVxEEB6EObvZx4">
15                      <img src="homes_635x250_1438606860.jpg">
16                  </a>
17              </div>
18              <div class="package-body">
19                  <a href="/mainStory?RS=ID_cGicsBVxEEB6EObvZx4">
20                      How American homes have changed since the '50s
21                  </a>
22              </div>
23          </div>
24          <div class="footer-section">
25              <span class="footer-img-holder" data="ID_cGicsBVxEEB6EObvZx4" hint="Click">
26                  <img src="homes_120x48_1438606860.jpg">
27              </span>
28              <span class="footer-img-holder" data="ID_cGicsBVxEEB6UObvZx4" hint="Click">
29                  <img src="mom-2_120x48_1438619986.jpg">
30              </span>
31              <span class="footer-img-holder" data="ID_cGicsBVxEEB6kObvZx4" hint="Click">
32                  <img src="school-beauty1_120x48_1438634270.jpg">
33              </span>
34              <span class="footer-img-holder" data="ID_cGicsBVxEEB60ObvZx4" hint="Click">
35                  <img src="1craig-james_120x48_1438633377.jpg">
36              </span>
37              <span class="footer-img-holder" data="ID_cGicsBVxEEB7AObvZx4" hint="Click">
38                  <img src="GOP-in-NH2ipad_120x48_1438658489.jpg">
39              </span>
40          </div>
41      </div>
42  </body>
```

Figure 2. The HTML Code of the Example



```
1   function updateMainStory(msg) {
2       $('#main_story_content').html(msg);
3   }
4   function showMainStory() {
5       var id = this.getAttribute('data');
6       var url = 'http://www.demo.org/main-story?id=' + id;
7       $.ajax({ url: url, success: updateMainStory });
8   }
9   function showHint(e) {
10      if (!$('#hint').length) {
11          $(document.body).append('<div id="hint"></div>');
12      }
13      $('#hint').html(this.getAttribute('hint'));
14      $('#hint').css({'display':'block', 'left':e.pageX, 'top':e.pageY});
15  }
16  function hideHint() {
17      $('#hint').css({'display':'none'});
18  }
19  $('.footer-img-holder').click(showMainStory);
20  $('.footer-img-holder').mouseover(showHint).mouseout(hideHint);
21  $(document).on('mouseover', '#key, #search-submit', showHint);
22  $(document).on('mouseout', '#key, #search-submit', hideHint);
```

Figure 3. The JavaScript Code of the Example

hot news item (*.footer-img-holder*), the text input control (*#key*), and the submit button (*#search-submit*). Note that the code register event handlers using event delegation mechanism instead of registering event handlers directly, in line 21 and 22, which register two event handlers to *document* element instead of the text input and submit button elements.

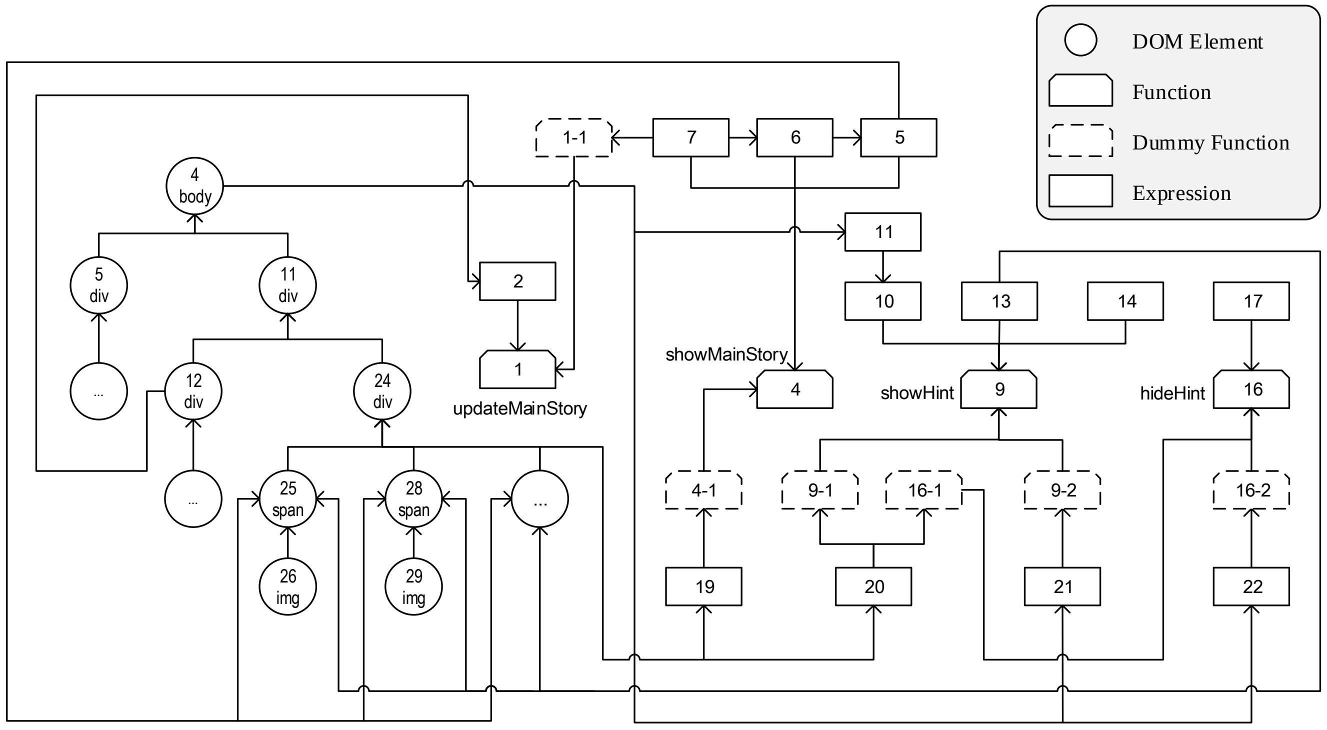


Figure 4. The Dependency Graph of the Example

Our example illustrates several important features of web applications. Web applications always use JavaScript libraries to ease the development, which make up a large percentage of all JavaScript code. Libraries always supply powerful APIs so that developers are able to do their jobs within a few lines of code, which causes a lot of details to be hidden in libraries. Lastly, a function can be registered as event handlers to multiple DOM elements at the same time, no matter whether they belong to the same web control or not.

3. Approach Overview

Our approach extracts web controls from web pages. Here, we define a web control is a subset of a web page, which satisfies: 1) it contains a set of visually and behaviorally distinct HTML elements, 2) it encapsulates structure and the behavior on that structure, 3) it is self-contained, i.e. it never interacts with other elements out of the control. Based on this definition, we can find that there are two web controls in the above example, i.e. the search engine entry and the hot news viewer. Note that the big picture viewer and the hot news list are not web controls, because they are not self-contained (they interact with each other).

Web pages are developed with a combination of three different languages: HTML, CSS and JavaScript. HTML is a markup language that defines structure and content of a web page. CSS is a style sheet language that expresses the presentation of web pages. JavaScript is a scripting language that defines the behavior of web pages. Elements of HTML form a tree structure, which is quite simple to extract. CSS is often used through the whole web sites, and we believe that it can be kept fully. JavaScript code is always composed of user code and libraries. Libraries are often used through the whole web sites, and just like CSS, there is no need to slice them. Different user code is developed for different web pages, and thus only user code that related to web controls should be extracted.

The life-time of the web pages can be divided into two steps: 1) page initialization, when the browser parses the web page code and builds the DOM of the page, and 2) event-handling, when users interact with the pages [16]. In the first step, JavaScript is used to build the DOM, including mutating DOM states, registering event handlers. In the second step, JavaScript is used as event handlers to interact with users.

To identify JavaScript code of a specified web control, we need to identify code that 1) mutates the web control, 2) defines event handlers of the web control and 3) registers these event handlers. We identify code that mutates the web control by monitoring the effect of executing JavaScript code on DOM state, identify code that defines event handlers by firing all events on the specified web control, which triggers execution of event handlers, and then record the executing functions and identify their definitions via the dependency graph. Finally, we identify code that registers these event handlers via the same dependency graph, the registration code should depend on the definition code.

In order to identify the three types of JavaScript code efficiently, we conduct a dynamic analysis on only user defined JavaScript code, which results in a dependency graph that cannot cover the full dependency relationships. In essence, the process of identification is performing dynamic program slicing on HTML and JavaScript code. To accomplish the process, we define four set of nodes as the slicing criterion: generates a selected-DOM-element set (S1), a sibling-of-selected-DOM-element set (S2), an executed-function set (S3), and a un-executed-function set (S4), where code depended on by S1 is the one that mutates the specified web control, code depended on by S3 is the one that defines event handlers of specified web control and code depended by S3 registers them, and S2 and S4 for other web controls as well. We perform the slicing by preserving code that is related to specified web control, discarding code that is related to other web controls, and preserving code having no relationship with these web controls.

Our approach is composed of two phases: execution and extraction. In execution phase, our approach monitors the execution of page initialization and event handlers registered on the web control. We require the user to point out the root DOM element of intended web control, and to interact with the control to trigger all event handlers registered on that control. We assume that the one who wants to extract a web control has the knowledge of the internal details of the web control. This is in line with the assumptions presented in [16] [17]. While the code is executing, we build a dependency graph of the DOM tree and executed JavaScript code, and generates the four sets as slicing criterion. In extraction phase, we perform program slicing based-on the dependency graph and the slicing criterion, and generate the HTML code and JavaScript code of the specified web control.

We illustrate the process of our approach via extracting the hot news viewer from above example.

Before the start of process, the user are required to point out user defined JavaScript code of a web page, which is the target of analysis in the next step.

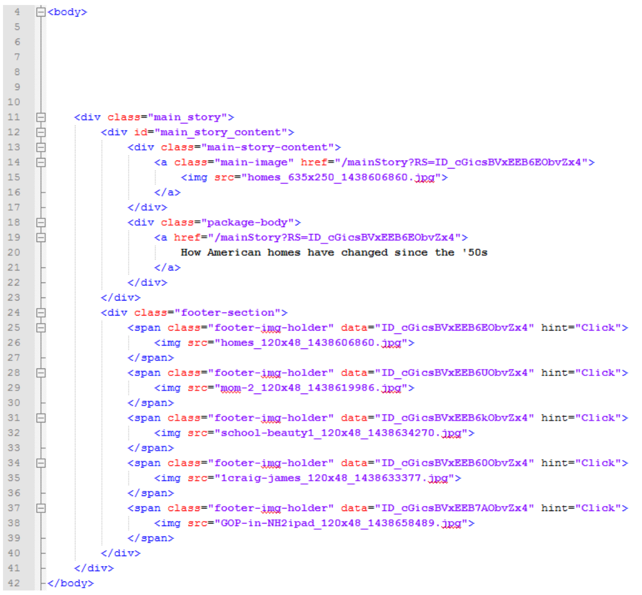
Firstly, in execution phase, user opens the web page in a browser, and the web page starts to initialize automatically. Then user gives out a XPath "*/html/body/div[2]*", which specifies root HTML element of hot news viewer, and triggers all the three event handlers on hot news viewer via moving on/out and clicking on each of five small pictures at the bottom. At the same time, our approach builds a dependency graph for executed JavaScript code and initialized DOM tree, as shown in Fig. 4, and generates four sets list as following:

- **S1** : selected-DOM-element set
  { 11, c_{11} | c_{11} is a child node of node 11 }
- **S2** : sibling-of-selected-DOM-element sets
  { 5, c_5 | c_5 is a child node of node 5 }
- **S3** : executed-event-handler set
  { 1-1, 4-1, 9-1, 16-1 }
- **S4** : un-executed-function set
  { 9-2, 16-2 }

Secondly, in the extraction phase, we traverse the dependency graph, and perform extraction by the following steps. For HTML code, we extract HTML elements in S1 and their parent elements, as shown in Fig. 5. For JavaScript code, we discard JavaScript code that depends on or is depended on by nodes in S2 and S4, and does not depend on or is not depended on by nodes in S1 and S3, i.e. node 21 and 22. Then we get the JavaScript code of intended web control, as shown in Fig. 6.

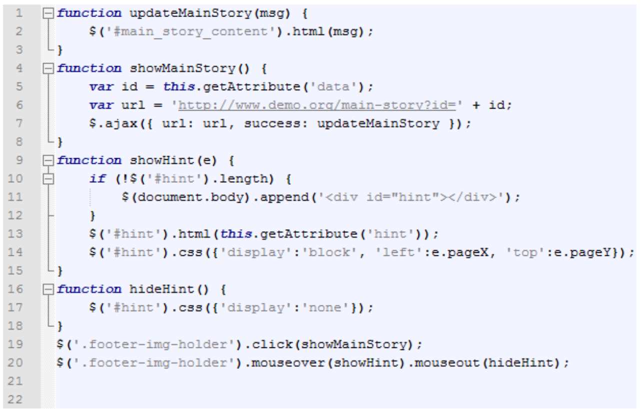
4. Approach

Our approach is composed of two phases, execution and extraction. Initially, we instrument the web page with the monitor code. Then, in the execution phase, our approach runs the instrumented web page in a web browser, along with an XPath specifying the root HTML element of intended web control, and a series of user actions triggering event handlers on the control. When the web page is running, our approach monitors the executed JavaScript code, generates a dependency graph and four node sets as slicing criterion as output. Finally, the extraction phase receives the dependency graph and the slicing criterion as input, performs slicing upon the source code of the web page, and generates the source code of intended web control as output. The overall process is shown in Fig. 7.



```
4   <body>
5
6
7
8
9
10
11      <div class="main_story">
12          <div id="main_story_content">
13              <div class="main-story-content">
14                  <a class="main-image" href="/mainStory?RS=ID_cGicsBVxEEB6EObvZx4">
15                      <img src="homes_635x250_1438606860.jpg">
16                  </a>
17              </div>
18              <div class="package-body">
19                  <a href="/mainStory?RS=ID_cGicsBVxEEB6EObvZx4">
20                      How American homes have changed since the '50s
21                  </a>
22              </div>
23          </div>
24          <div class="footer-section">
25              <span class="footer-img-holder" data="ID_cGicsBVxEEB6EObvZx4" hint="Click">
26                  <img src="homes_120x48_1438606860.jpg">
27              </span>
28              <span class="footer-img-holder" data="ID_cGicsBVxEEB6UObvZx4" hint="Click">
29                  <img src="mom-2_120x48_1438619986.jpg">
30              </span>
31              <span class="footer-img-holder" data="ID_cGicsBVxEEB6kObvZx4" hint="Click">
32                  <img src="school-beauty1_120x48_1438634270.jpg">
33              </span>
34              <span class="footer-img-holder" data="ID_cGicsBVxEEB60ObvZx4" hint="Click">
35                  <img src="1craig-james_120x48_1438633377.jpg">
36              </span>
37              <span class="footer-img-holder" data="ID_cGicsBVxEEB7AObvZx4" hint="Click">
38                  <img src="GOP-in-NH2ipad_120x48_1438658489.jpg">
39              </span>
40          </div>
41      </div>
42  </body>
```

Figure 5. The Extracted HTML Code



```
1   function updateMainStory(msg) {
2       $('#main_story_content').html(msg);
3   }
4   function showMainStory() {
5       var id = this.getAttribute('data');
6       var url = 'http://www.demo.org/main-story?id=' + id;
7       $.ajax({ url: url, success: updateMainStory });
8   }
9   function showHint(e) {
10      if (!$('#hint').length) {
11          $(document.body).append('<div id="hint"></div>');
12      }
13      $('#hint').html(this.getAttribute('hint'));
14      $('#hint').css({'display':'block', 'left':e.pageX, 'top':e.pageY});
15  }
16  function hideHint() {
17      $('#hint').css({'display':'none'});
18  }
19  $('.footer-img-holder').click(showMainStory);
20  $('.footer-img-holder').mouseover(showHint).mouseout(hideHint);
21
22
```

Figure 6. The Extracted JavaScript Code

4.1 The Execution Phase

This phase monitors the execution of JavaScript code, and generates a dependency graph and slicing criterion for the next phase.

As mentioned above, the lifetime of a web page can be divided into two steps, page initialization and event-handling. In page initialization step, all initialization code executes, no matter whether it is related to the specified web control or not, we have to separate related code from others; while in event-handling step, only event handlers of specified web control are triggered, and functions executed in this step should be extracted indiscriminately. We separate initialization code related to the

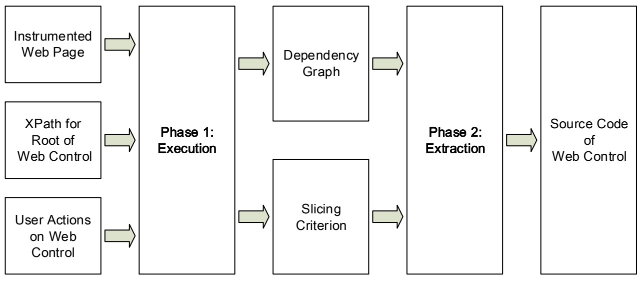


Figure 7. The Process of Our Approach

specified web control based on the dependency relationships between initialization code and executed event handlers.

We build a dependency graph to describe the relationships between event handlers and initialization code. When the JavaScript is executing, our approach records expressions that mutate DOM elements within and outside the specified web control, functions that ever and never run, as the slicing criterion.

4.1.1 The Dependency Graph

The dependency graph represents the dependency relationships between DOM elements and JavaScript expressions. We introduce two types of nodes, DOM node and JavaScript node, to represent DOM elements and JavaScript expressions respectively, and four types of edges to represent relationships:

- A DOM-to-DOM edge represents a DOM element depending on another DOM element, i.e. a child DOM element depends on its parent DOM element.
- A DOM-to-JavaScript edge represents a DOM element depending on a piece of JavaScript expression, i.e. a DOM element depends on a piece of JavaScript expression which writes on it, including attribute assignment and some method invocations.
- A JavaScript-to-JavaScript edge represents a piece of JavaScript expression depending on another piece of JavaScript expression, there are several cases: 1) an expression reading a value depends on the expression set the value, 2) an expression inside a compounded expression depends on the compounded expression, such as for, if, etc., 3) an invoked function depends on the expression that invokes it.
- A JavaScript-to-DOM edge represents a piece of JavaScript expression depending on a DOM element, i.e. a piece of JavaScript expression reads values from a DOM element, including values of attributes and return values of a method invocations.

Additionally, we divide JavaScript nodes into three sub-types: function node, dummy function node, and expression node. Because we perform slicing based on whether a function is an event handler of intended web control, we define function node to distinguish function expressions from ordinary expressions. Besides, a function may be registered as event handlers on elements within or outside the intended web control at the same time (e.g., the *showHint* and *hideHint* are registered to the search engine entry and the hot news viewer in the above example). To distinguish functions that may be registered to different DOM elements as event handlers, we create a dummy function for each function that may be registered as an event handler, i.e. passed as a parameter to another function call. We define a dummy function node to represent a dummy function.

4.1.2 The Slicing Criterion

As mentioned above, in order to extract web control efficiently, we perform dynamical analysis on only user defined JavaScript code, which causes we cannot get a dependency graph that covers the full dependency relationships.

We introduce a hybrid method to perform program slicing based on the incomplete dependency graph. Based on this dependency graph, simply extracting code related to the intended web control, or simply discarding code related to other DOM elements, may not generate an executable web control. Therefore, our approach not only identifies code related to intended web control, which should be preserved, but also identify code related to other DOM elements, which should be discarded Furthermore, code related to both should be preserved because the intended web control does depend on it, and code related to neither should be preserved because the intended web control may depend on it.

In detail, we define four sets as slicing criterion: DOM elements within the intended web control (S1), sibling elements of intended web control (S2), event handlers of the intended web control (S3), and un-executed functions (S4), where S1 and S3 are used to identify code related to intended web control, S2 and S4 are used to identify code related other DOM elements.

We generate S1 and S3 by checking the relationship between each DOM element and the root element of intended web control recursively. Initially, we put the root element into S1, and set S3 to empty; then we traverse the DOM tree recursively and check: if an element is a child of the root element, we put it into S1, else if it is neither a child nor a parent of the root element, we put it into S3.

We generate S2 and S4 by checking whether a dummy function is executed in event-handling phase. Initially, we set S2 to empty, and put all functions into S4; in event-handling phase, when we monitor a dummy function is executed, we remove it from S4 and put it into S2.

4.2 The Extraction Phase

This phase perform slicing base on the incomplete dependency graph and the slicing criterion generated in the execution phase. We preserve all CSS code and JavaScript libraries pointed out by users, only extract code from HTML and user defined JavaScript code.

4.2.1 Extracting HTML Code

We extract DOM elements within the intended web control and their parent elements via the following steps: 1) we extract all elements in S1, 2) we randomly select an element from S1 (all elements in S1 are under the same sub DOM tree), traverse the path following the element's parent chain upward, and extract each encountered DOM element.

4.2.2 Extracting JavaScript Code

We extract JavaScript code via the follow steps: 1) we mark on each node of dependency graph to represent whether it should be preserved or discarded; 2) we discard nodes marked with being discarded only.

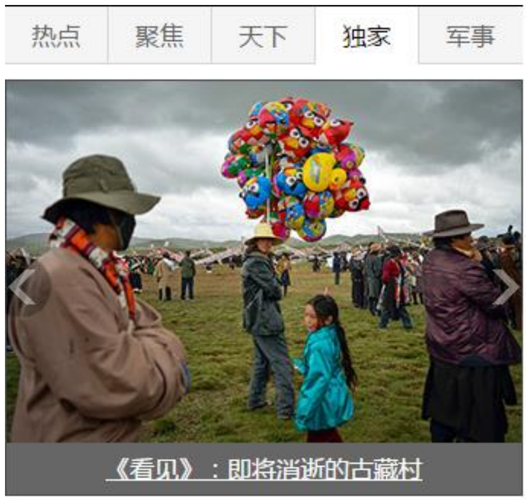


Figure 8. Focus News from Sina.com

Marking Graph

We pick up a node from S1, S2, S3, and S4 sequentially, and mark the graph as follows:

- For a node *n* from S1, we mark each succeed node of *n* with a flag *P* recursively, which represents that nodes depending on *n* should be preserved;
- For a node *n* from S2, we mark each succeed node of *n* with a flag *D* recursively, which represents that nodes depending on *n* should be discarded;
- For a node *n* form S3, we mark *n*, each succeed and previous node of *n* with a flag *P* recursively, which represents that *n* and nodes depending on or depended on by *n* should be preserved;
- For a node *n* form S4, we mark *n*, each succeed and previous node of *n* with a flag *D* recursively, which represents that *n* and nodes depending on or depended on by *n* should be discarded.

Discarding Code

After the dependency graph is marked, we traverse the dependency graph and discard nodes that are marked with *D* only, which means JavaScript code of these nodes is determined to have nothing to do with the intended web control totally.

To keep the executability of extracted JavaScript code, an additional work is to fix expressions that depend on discarded functions. Because the discarded functions are never invoked during event-handling phase of extracted web control, we replace the reference to a discarded function with an empty artifact function within a preserved expression, to avoid the extracted web control to raise an "Uncaught Reference Error" in the future.

5. Evaluation

We performed two case studies using real-world web applications. We extracted two web controls from two popular web applications using our approach, and manually checked whether each extracted web control presented and acted the same as that in the context of original web page. We try to answer the following research questions:

RQ1: Is our approach able to extract web controls executing correctly?

Table 1 Comparing Result of Sina News

	Original	Extracted	Save
Code Size	1,122 KB	252 KB	78 %
Resource Size	19,586 KB	195 KB	99 %

RQ2: What is the effect of our approach?

RQ3: How efficient can our approach extract a web control?

5.1 Case Study 1

Our first web control came from sina.com [3]. Sina.com is the largest Chinese-Language web portal, which provides a lot of feature news as the focus of the lead story or the main photo. We extracted the focus news control from the home page of news center of sina.com, http://news.sina.com.cn.

The focus news control is composed of four components: a tab at the top, an image viewer at the bottom, a pair of arrows at two sides, and a switcher without presentation, as shown in Fig. 8. The tab is composed of five items, mouse moving over which causes switch of images displayed in the image viewer. The image viewer periodically displays one image out of five, which is under control of the switcher. Lastly, the pair of arrows allows users to switch images manually, by clicking one of the two arrows.

Before extraction, we downloaded the source code of subject web page to local storage. The web page could not execute correctly locally, because it dynamically loaded external JavaScript code from remote server. Moreover, it encrypted the core of remote code load library. So we modified and simplified the JavaScript code of that page, to ease the manual work in the future. Firstly, we separated user code from libraries and advertisements manually, and instrumented user code by Jalangi [19]. Then we started the extraction process by giving out the XPath of the focus news control, *//*[@id="wrap"]/div[8]/div[1]/div[1]*, and performed the following actions on the control: firstly, we waited for the image viewer automatic cycling, then we moved the mouse over each of the five items of the tab, finally we clicked the left and right arrows per five times. Then the control is extracted, and we run the extracted control and compared it with that in the context of the original page. We did these works on a Google Chrome browser, on a personal computer with Intel Core i7 CPU 3.40 GHz processer, 8GB of memory.

***RQ1*:** we evaluated the correctness of extracted focus news by checking: 1) whether it presents the same as in the context of the original web page, 2) whether it acts the same as in the context of the original web page. We manually compared the presentations of the extracted one with the one in the original web page, and they looked the same exactly. Then we performed the same actions as doing in extracting phase on the extracted control, and it played just the same as before. So we believe we extracted the focus news control correctly.

RQ2: To measure the effect of our approach on focus news control, we counted the code size and resource size of extracted control, and compared them with the ones before extracting. Table 1 shows the result. The code includes HTML code, CSS code, and JavaScript code, while the resource includes images besides code. From the table we can find that our approach deduces the code size and resource size of the focus news control by 78% and 99%, and deduced codes are mainly HTML code out of intended web

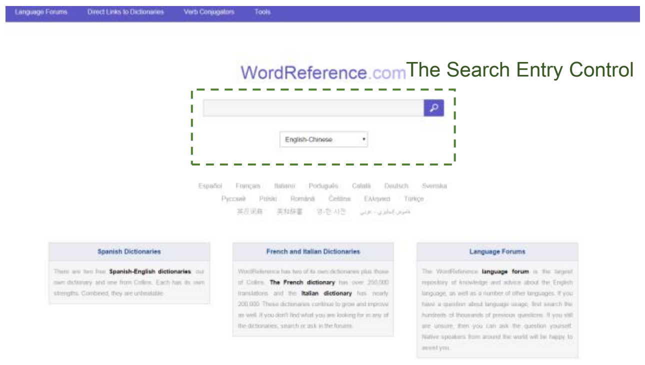


Figure 9. Home Page of WordReference

control, JavaScript code of advertisements and other controls. Deduced resources are mainly images of other controls. The result shows fewer computation and bandwidth is needed to visit focus news of Sina via our approach, which means great energy and bandwidth saving.

RQ3: To measure the efficiency of our approach, we performed the extraction process for ten times, and computed the average time cost on extraction phase. We found that it took from 0.77s to 0.86s, with an average 0.82s, to finish the extracting process. Compared with our approach, FireCrow [16], which is another client side extracting tool, was not able to finish the process within 10 minutes. Therefore, we believe that our approach is able to extract a web control from a rich web page like sina efficiently.

5.2 Case Study 2

Our second web control came from WordReference [20]. WordReference is the most famous online translation dictionary for multiple languages. The home page of WordReference is very simple, it defines only two web controls, and uses only an external image less than 1 KB as a search icon, and no external CSS file. The two web controls are a navigator at the top and a search entry in the middle, as shown in Fig. 9. Our goal is to extract the search entry control in the middle.

The search entry control is composed of four components: a text input, a submit button, a selector, and a word recommender. Users are allowed to type a searching word via the text input, and submit the search request by clicking the submit button. When users are typing a character, the word recommender appears showing words starting with user typed characters. Lastly, users are allowed to switch source and destine language for translating by changing the value of the selector.

We performed the extraction process following the same steps as that had been done in case 1. We downloaded the source code to local storage, identified user defined JavaScript code manually, and instrumented it by Jalangi. Then we gave out the XPath, *//*[@id="text_form"]*, and performed the following actions: firstly, we moved the mouse over the input control, then typed ‘a’ two times, finally, we changed the value of the selector to “Chinese-English”. Then we run the extracted control and compared it with that in the context of the original page. We did these works in the same environment as case 1.

***RQ1*:** we evaluated the correctness from representation and behavior respects separately. Firstly, we compared the representation of extracted search entry control with the one in the

Table 2 Comparing Result of WordReference

	Original	**Extracted**	**Save**
Code Size	225 KB	173 KB	23 %
Resource Size	226 KB	174 KB	23 %

original page, and found it represented exactly the same as that before extracting, not only the appearance, but also the position where it displayed. Secondly, we submitted ten requests for searching “aa” under different language settings through the extracted search entry, and found it submitted the requests to the same pages as doing through the original page, except the word recommender didn’t appear on the extracted one. This is because the appearance of recommender is triggered by the callback function of an asynchronous request, while executing the extracted control locally is forbidden to submit a synchronous request to a remote server because of the Same-Origin-Policy [21]. However, we analyzed the source code and found that all code required by the control was preserved correctly, so we believe that it would work correctly when deployed at the same server of original page.

RQ2: Table 2 shows the effect of our approach on the search entry control, comparing the code size and resource size of extracted control with the ones before extracting. The code includes HTML code and JavaScript code only, while the resource includes one image less than 1 KB besides code. From the table we can find that our approach deduces both the code and resource size by 23%. The result shows even for an almost the simplest web page, our approach is able to deduce cost on energy and bandwidth.

RQ3: As done in case 1, we computed the average time cost on extracting the search entry control for ten times. We found it took from 0.14s to 0.17s, with an average 0.15s, to finish the extracting process, compared with FireCrow [16], which took about 4 seconds. So we believe that our approach is able to extract a web control from a simple web page such as WordReference efficiently.

5.3 Threads to Validity

Our evaluation is performed on only two web applications. However, these web applications considered are selected from Alexa top sites, of different categories, and of different sizes. They are representative in web application mobilization. Further, we manually checked the correctness of extracted web controls, which may be unsound. But, we have done our best. In future work, we will evaluate our approach on more web applications of more categories.

6. Discussion

Generally, developers use a CSS selector to locate DOM elements, which is often composed of a single element id, or a list of style rules representing the contextual information. From CSS Level 3 [21], developers are allowed to locate an element by the sequence number of siblings. However, our approach extracts only the elements within intended web control and its parent elements, and discards its parent’s siblings, which destroys the structure of the DOM tree, and may invalidate the selectors composed of sequence number of siblings. We find cases like that seldom exist, and we will handle them in the future.

Our approach extracts event handlers only triggered by user actions performed in execution phase. Currently we perform the execution and extraction on a computer, and can only trigger traditional desktop event handlers, such as MouseEvent and KeyboardEvent. So the extracted web controls don't match mobile devices well. To extract web controls exactly match mobile devices on a desktop device is another work in the future.

7. Related Works

User Interface Adaptation. Many works use user interface adaptation to fit traditional desktop web pages into small displays for mobile devices. Huang et al. [12] detect blocks fitting a given size of screen in a runtime web page by their actual width and height. Chen et al. [1] detect blocks by explicit or implicit separators of a web page. These are vision based page segmentation works. Hua et al. [10] identify semantic blocks based on the observation that semantics blocks are designed with distinct themes and are distinguishable from the other parts of a dynamic page, which is a content based page segmentation work. These works focus on only the problem of different screen size, by detecting blocks from a whole web page automatically and locating to the block that users may be interested in, but users are still forced to download the whole web page. While our approach focus on problems causes by huge resource size besides caused by small screen size, with the assist of our approach, users can not only get fitting views, but also save their bandwidth and power.

Web Control Extraction. Maras et al. presents a series of works to extract web controls from web pages for programming reusing [16] [23] [24] [25]. It is similar to our approach that they use dynamic analysis to extract related code of a web control from executed JavaScript code. The difference is they concern the accurate of extracted code, they analysis all executed code including large scale of libraries, and perform a fine-grained extraction on each expression, which results their approach hardly handles real-world web applications of large scale of JavaScript code. While our approach concerns scalability, we analysis only user defined JavaScript code, and extracts code based on function definitions.

8. Conclusion

Traditional web applications are not suitable for accessing via mobile devices directly. We propose an approach that is able to extract a part of web page efficiently. Our approach monitors the execution of functions of initialization and event handlings of the web control, and performs extraction by discarding code that doesn't depend on or is depended on by the intended web control. Two case studies on real-world web applications show that our approach is able to extract web controls correctly and efficiently.

9. REFERENCES

[1] Y. Chen, WY. Ma, and HJ. Zhang. Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices, In Proc. World Wide Web, 2003

[2] http://zeendo.com/info/mobile-screen-sizes

[3] SinaNews. http://news.sina.com.cn

[4] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making Smartphones Last Longer with Code Offload, in The 8th Annual International Conference on Mobile Systems, Applications, and Services, 2010

[5] SinaWap. http://sina.cn/?from=wap

[6] SiteApp. http://zhanzhang.baidu.com/tools/siteapp

[7] Baidu. http://www.baidu.com

[8] IBM. http://www.ibm.com

[9] Worklight. http://www.ibm.com/software/products/en/mobilefirstfoundation

[10] Zg. Hua, X. Xie, H, Liu, Hq. Lu, and Wy. Ma. Design and Performance Studies of an Adaptive Scheme for Serving Dynamic Web Content in a Mobile Computing Environment. In IEEE Transactions on Mobile Computing, 2006

[11] Xy. Xiao, Q. Luo, D. Hong, HB. Fu, X. Xie, and Wy. Ma. Browsing on Small Displays by Transforming Web Pages into Hierarchically Structured Subpages. In ACM Transactions on the Web, 2009

[12] G. Huang, and Dm. Wang. Adapting User Interface of Service-Oriented Rich Client to Mobile Phones. In Proceedings of The 6th IEEE International Symposium on Service Oriented System Engineering, 2011

[13] G. Li, E. Andreasen, and I. Ghosh. SymJS: Automatic Symbolic Testing of JavaScript Web Applications. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014

[14] Yahoo. Https://www.yahoo.com

[15] jQuery. http://www.jquery.com

[16] J. Maras, M. Stula, J. Carlson, and I. Crnkovic. Identifying Code of Individual Features in Client-side Web Applications. In IEEE Transactions on Software Engineering, 2011

[17] T. Eisenbarth, and R. Koschke. Locating Features in Source Code. In IEEE Transactions on Software Engineering, 2003

[18] Alexa. http://www.alexa.com/topsites

[19] K. Sen, S. Kalasapur, T. Brutch, and S. Gibbs. Jalangi: A Selective Record-Replay and Dynamic Analysis Framework for JavaScript. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, 2013

[20] WordReference. http://www.wordreference.com/

[21] Same Origin Policy. http://www.w3.org/Security/wiki/Same_Origin_Policy

[22] CSS Selector Level 3. http://www.w3.org/TR/2009/PR-css3-selectors-20091215/

[23] J. Maras, M. Stula, and J. Carlson. Reusing Web Application User-Interface Controls. In The 11th International Conference on Web Engineering, 2011

[24] J. Maras, J. Carlson, and I. Crnkovic. Extracting Client-side Web Application Code. In The World Wide Web Conference, 2012

[25] J. Maras, J. Carlson, and I. Crnkovic. Towards Automatic Client-side Feature Reuse. In The 14th International Conference on Web Information System Engineering, 2013