PrintMonkey: Giving Users a Grip on Printing the Web

Jennifer Baldwin
University of Victoria
3800 Finnerty Rd.
Victoria, BC, Canada
jbaldwin@cs.uvic.ca

James A. Rowson
Hewlett-Packard Laboratories
1501 Page Mill Rd.
Palo Alto, CA, USA 94304
jim.rowson@hp.com

Yvonne Coady
University of Victoria
3800 Finnerty Rd.
Victoria, BC, Canada
ycoady@cs.uvic.ca

ABSTRACT

Web content is notoriously difficult to capture on a printed page due to inconsistent and undesired results. Items that users may not want to print, such as media, navigation menus and more show up on their page. Other items that they may care about are truncated or spread across several pages. Some tools exist to help users with what is printed, but they often are cumbersome to use or are costly for a company to maintain. Therefore, we introduce PrintMonkey, which allows users to write their own printing templates and share them with others on the web. No modifications to the original webpages are required and users with less development experience can use and develop templates. A comparison with four alternative solutions reveals the concrete ways in which PrintMonkey improves upon existing approaches in terms of functionality, customizability and scalability.

Categories and Subject Descriptors

I.7.2 [**Document and Text Processing**]: Document Preparation—*hypertext/hypermedia, languages and systems, markup languages, scripting languages*

General Terms

Design, Experimentation, Algorithms, Human Factors

Keywords

Printing the web, print templates, customized browsing, screen scraping, JavaScript

1. INTRODUCTION

It has recently been reported that printing is one of the most used features in a browser and it is highly likely that more than half of the pages printed are from web content [9]. However, printing webpages is often frustrating for users because the printed output does not match what they see in the browser. Often there are too many pages generated, many of which are mostly blank, poor page layout and advertisements. For example, the New York Times website looks normal in a browser, as shown in the top part of Figure 1, but an attempt to actually transfer this to a printed page produces the output shown in the bottom part of Figure 1.

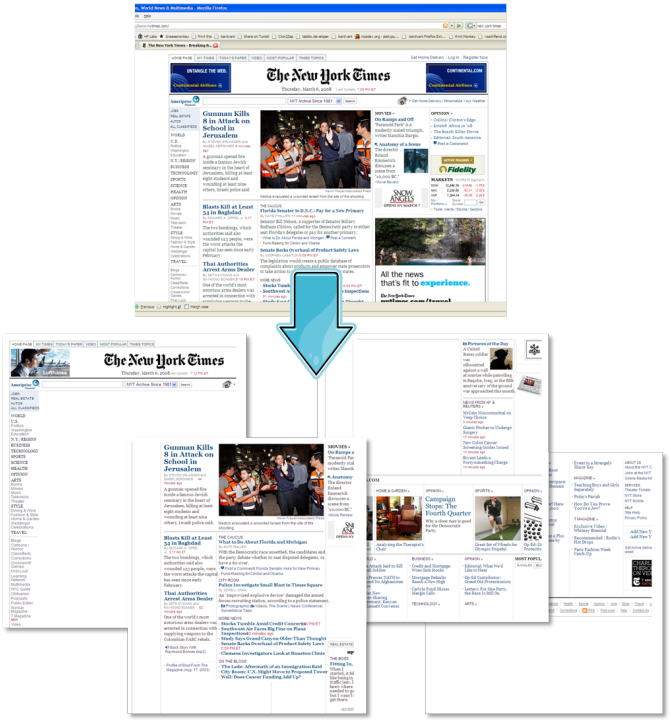

Figure 1: Viewing and Printing New York Times

Tools already exist to customize viewing pages in a browser. Adapting pages for use with mobile devices is a very active area of research [17, 23, 24, 20]. Firefox [4] can use Greasemonkey [7, 22] or related technologies [19, 15]. Greasemonkey is a Firefox extension that allows *user scripts* to run on a specific webpage. These user scripts are simply balls of JavaScript [11] that alter the way a page appears or even the way a page is used. For example, popular scripts include those that customize a GMail layout, including its color scheme, or even embedding price comparison in Amazon webpages. If this were adapted to printing, then the printed output would match what is seen on the screen. However, these Greasemonkey scripts will run each and every time the page is viewed, so this does not quite suit every day users' needs in that they may want to print something differently than how they view it.

PrintMonkey allows users to modify how pages are viewed in the

DocEng'08, September 16–19, 2008, São Paulo, Brazil.

same way as Greasemonkey but instead for printing. For example, to print a GMail inbox, we could eradicate colors and have snippets of the emails. PrintMonkey is as easy to use as Greasemonkey, but has the additional feature that PrintMonkey scripts are more accessible to users in terms of development – in particular for users with minimal programming background. We propose a strategy where users are also able to share templates with each other, and every time a template is selected, its popularity increases so templates will be better targeted for specific sites.

This paper is organized as follows. The rest of this section introduces related work and analyzes strengths and weaknesses of existing tools. Section 2 outlines the requirements for PrintMonkey, its user interface and some implementation challenges. In Section 3, we discuss how developers can create different printing templates and Section 4 provides an evaluation of PrintMonkey relative to current approaches. Future work is introduced in Section 5 and we conclude in Section 6.

1.1 Related Work

There are four current approaches for printing the web that we are aware of. These include built-in web browser support, HP Smart Web Printing [8], Cascading Style Sheets [2] and the Tabblo Print Toolkit [16].

It is important to note that web browsers are now taking into account the lack of printing support. Internet Explorer 7 [10] has provided special capabilities for printing such as shrinking text so that all of the content fits on the printed page. The user can also remove headers/footers, edit margin sizes, customize page layouts and change the print space. These features are extremely useful but do not take into account items the user does not want to print on the page and does not allow them to save how they print that page for future use. Firefox also allows users to do many of the same things such as shrink to fit, landscape/portrait modes and setting the margins.

HP Smart Web Printing lets the user select, store and organize text and graphics from multiple webpages and print exactly what is seen on the screen. The problem with this approach is that it requires a lot of manual effort from the user in order to create something they only print once. They cannot save the templates for reuse, and further, cannot share their printing templates with others. However, it does allow the user to collect from multiple pages without any templates or programming knowledge.

Web developers can also alleviate printing woes with the use of cascading style-sheets (CSS). Most web developers already use CSS to set the appearance of the webpage in the browser. But developers can include multiple style sheets within a single webpage, so that one can control the appearance in the browser and another the way the page is printed. For example, with CSS, you can hide advertisements, banners, navigation bars, as well as change font size and style. You could also add images to the printout such as a copyright notice. If web developers consciously provided these style sheets, users' web printing experiences would be improved. However, most web developers are not concerned with how the page prints so much as how it looks and are not willing to go to the extra effort. Users may specify their own CSS through the browser but this also requires them to create it themselves.

The Tabblo Print Toolkit (TPT) is a suite of developer tools for making websites more printable. The owner of the site must edit their HTML to identify the content on the page that should be printed, using JavaScript. This is similar to a print CSS file. When someone wants to print the page, TPT will combine the data for printing with pre-existing templates and generate a PDF file on the TPT server. The two main problems we see with this approach are that the original websites must be modified to incorporate TPT, and there are costs associated with running the server that generates these PDF files. On the other hand, TPT does allow the user to have absolute control over how the page is printed because it generates PDF. This PDF will print the same as seen on the screen and the same no matter which platform is used.

2. PROPOSED SOLUTION

Taking into account some of the shortcomings associated with the this survey of state-of-the-art solutions, we suggest an alternative set of requirements for a web printing tool. These requirements are:

- Runs primarily on the client with JavaScript
- User Friendly
- Non Invasive

To alleviate expenses associated with running a server to do the processing of pages before they are sent to the printer, we require that PrintMonkey[1], the name of our prototype implementation, run primarily on the client. We do this by running JavaScript in the user's browser. We must still store templates on a server but in this way, processing is kept to a minimum.

We also want PrintMonkey to be easily used. PrintMonkey has been designed for many different kinds of users. It is simple to use for the user who simply wants to print a webpage and it is easy to create templates for both experienced developers and the average developer. Existing challenges for non-experienced developers to write templates include that they may not know JavaScript or programming languages, in general, that well. JavaScript can be confusing even for those with experience. Therefore, PrintMonkey provides a copy and paste mechanism that we believe will be less of a challenge for users. We will discuss how we believe we have an effective solution for this potential obstacle to adoption in Section 3.3.

Finally, we want to ensure that people will actually use PrintMonkey. First and foremost is the fact that developers will not need to modify their original site in order for PrintMonkey to be applied with it. In addition, users can create a template once and reuse it as often as they wish, and these templates will automatically be shared with others. They also do not need to install a standalone program, only a few lightweight items into their browser.

2.1 Prototype Implementation

This section describes how a user would actually use PrintMonkey in order to print a page. We also discuss the design challenges we faced when deciding how to implement PrintMonkey.

2.1.1 User Interface

In order to print with PrintMonkey, the user will click the bookmarklet in their toolbar, shown in Figure 2. This bookmarklet is a small JavaScript program that is stored in a URL bookmark.

Once they have clicked the bookmarklet, another page will appear that shows the most popular printing templates for that particular URL, shown in Figure 3. The user can then select whichever template they prefer or they can elect to view more templates online. The icons shown in this figure are samples and not actual renderings of what the user will see when they print the page. When they select a template, another page will popup and the page will be printed, as shown in Figure 4.

[1]PrintMonkey is freely available at *http://www.printmonkey.org*

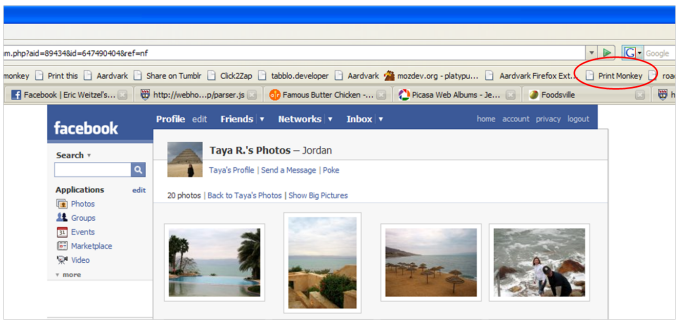

Figure 2: PrintMonkey Bookmarklet

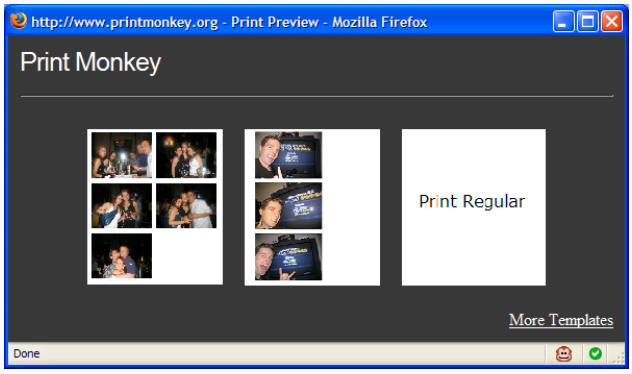

Figure 3: Most Popular Printing Options

2.1.2 *Prototype Implementation*

The most challenging part of developing PrintMonkey was getting around the cross-site scripting limitations. Cross-site scripting refers to the gathering of data from one site to another [21]. It is generally referred to as being malicious. For example, if you had your bank account information open in one browser window, you would not want to have a bookmarklet or link execute some JavaScript that reads the information contained within it.

In our case, we had the URL of the page we wanted to print and needed to get the HTML content from it. Since our code resides on a server on a different domain than pages we want to print, this was a security restriction. There were a couple of ways we could get around this. The first was to use a server side script to fetch the contents, such as PHP. However our requirements were that PrintMonkey should run mainly on the client-side and this would have created additional complexity for pages in which the user was logged in. Therefore this was not an option. We could have also used Internet Explorer with an ActiveX control. In the end, we chose to use Greasemonkey to fetch the contents for us because we wanted to work on the Firefox platform and believed Greasemonkey to be widely accepted.

Greasemonkey has a builtin method called `GM_xmlhttpRequest` which allows user scripts to get data from any URL. We created a Greasemonkey script that runs on our output print webpages and merely injects the HTML from the original page into the printable page so that it can be manipulated by printing templates. Since Greasemonkey is a Firefox plugin, this creates no issues with cross-site scripting. Of course, it is possible that this could be used maliciously if you attempt to print your bank software with a malicious PrintMonkey script. This also means that Greasemonkey currently

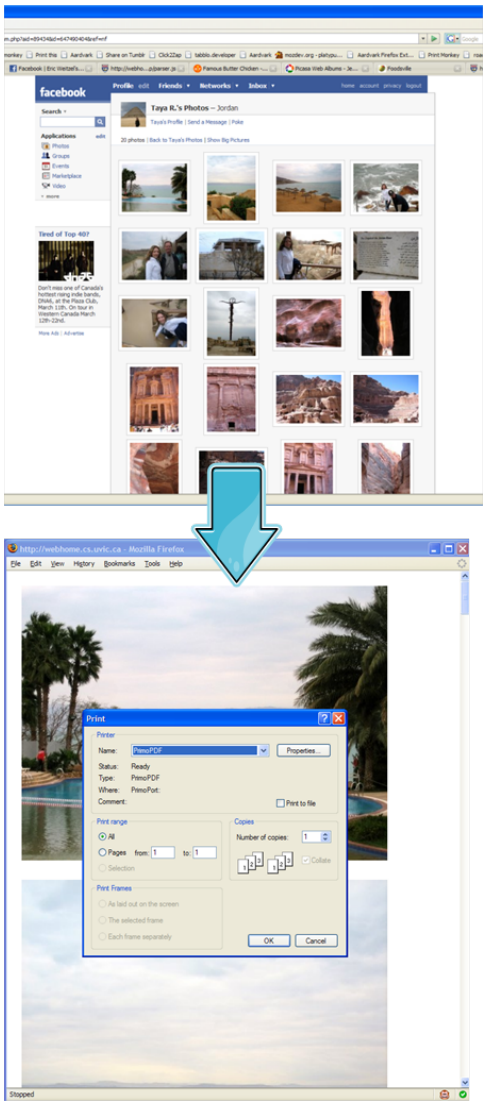

Figure 4: Final Printed Page

only runs on Firefox although in the future, we would like to have it run on Internet Explorer as well.

When we were building PrintMonkey, we also had to keep in mind that we wanted users to be able to write scripts for PrintMonkey so we needed to design for extensibility. How templates are actually used within a page are discussed in the following sections, along with the required meta-data for these scripts.

3. DEVELOPING FOR PRINTMONKEY

There are three types of templates that PrintMonkey understands. These are:

- Greasemonkey Style Templates
- JavaScript Templates
- PrintMonkey Templates

Greasemonkey and JavaScript templates are meant for the experienced developer who wants to maintain ultimate control over

the final printed product. However, JavaScript can be confusing, especially for the average developer. Therefore we provide an additional type of template for the inexperienced developer that we call PrintMonkey templates. Each of these templates are discussed in order in the following subsections. Further, proposed solutions to issues of sharing and debugging common to all scripts are examined in the final subsections, before an evaluation of PrintMonkey in Section 4.

3.1 Greasemonkey Style Templates

A Greasemonkey user script is a chunk of JavaScript code. Part of this script tells Greasemonkey on what pages it will run, using a regular expression which matches some set of URLs. Greasemonkey also provides special functions that are only available to user scripts. We have decided to support Greasemonkey style scripts with PrintMonkey because hundreds of them already exist in the Greasemonkey script repository. Some of these may not be applicable to printing, but those that are allow us to quickly adopt efforts that have already been completed. We say Greasemonkey "style" scripts because we mean scripts that are one large chunk of JavaScript. If special Greasemonkey functions are used, we cannot understand them because we run the script within the context of the page and not within the Firefox plugin itself.

Figure 5 shows an example of a Greasemonkey style script that converts a Facebook album page to a page containing the full size images from that page. The output was shown previously in Figure 4. It is important to note that it is mostly generic JavaScript, potentially confusing for a novice, with some PrintMonkey method calls thrown in. These additional methods PrintMonkey provides are:

- clearHead(): clears all previous style information from the page
- addToHead(): adds style information to the printable page
- loadResults(): loads an HTML string to the printable page
- disableLinks(): disables links on the printable page so that users cannot navigate away from it
- getURLParam(): retrieves parameters from the URL

We have provided these functions since they will probably be used often and will reduce mundane effort for the template developers.

3.2 JavaScript Templates

Greasemonkey style templates are also written in JavaScript as described above. The difference between what we refer to as JavaScript templates and Greasemonkey style templates are that JavaScript templates are split up into data mining scripts and the scripts which display that data, or what we call *visual scripts*.

The impetus for separating the two is that a single page will contain data that can be displayed in many ways. Additionally, a printed page may appear the same no matter which website the data was extracted from. To make this more concrete, consider an example where users are printing recipes found from online recipe sites such as Foodsville [5] or AllRecipes [1]. Both of these sites provide the same basic recipe information such as title, ingredients, steps and image. Therefore, each of these sites will have separate data mining scripts to retrieve the same recipe information. This recipe information could then be printed in a number of ways. When the printable page is actually rendered by PrintMonkey, the data script and selected visual script will be embedded in the same page so that the visual script will see all of the variables that the data script provides.

```
clearHead();
var album = document.getElementById("album");
var images = album.getElementsByTagName("img");
var newHTML = "<table cellpadding='15'>";

for(var index=0; index<images.length; index++) {
     var imagePath = images[index].attributes
                          .getNamedItem("src").value;

     //need to get big version
     var tokenArray = imagePath.split("/");
     var fileName = tokenArray[tokenArray.length -1];
     fileName = "n" + fileName.substr(1);
     var newName = "";

     for(var i=0; i<tokenArray.length-1; i++) {
          newName += tokenArray[i] + "/";
     }

     newName += fileName;
     newHTML += "<tr><td><img src='" + newName + "' />
                    </td></tr>";
}

newHTML += "</table>";
loadResults(newHTML);
```

Figure 5: Greasemonkey Script for Facebook

3.2.1 Data Scripts

The purpose of a JavaScript data script is simply to set up and populate the variables that are used by the visual script. It merely has to define variables the same way that JavaScript does (i.e. `var name =`).

3.2.2 Visual Scripts

A JavaScript visual script can use the same variables as defined in the data script as if they were already defined in this script itself. This is because, as mentioned previously, both scripts are simply embedded in the same page. This means that the names in each script are extremely important and cannot be mistyped. It is essentially a contract that each data script must provide variables with the same names as those used in the visual script and vice-versa. Similar to the Greasemonkey style scripts, the same PrintMonkey helper functions can be used in JavaScript templates as well.

3.3 PrintMonkey Templates

PrintMonkey templates are written in a language similar to HTML. Like JavaScript templates, they are also split into data gathering scripts and visual scripts. When these templates are used, they are converted to JavaScript and then inserted into the same page together. This means that similar to the JavaScript templates, the variables defined in the data script are the only ones that can be used in the visual script.

3.3.1 Data Scripts

PrintMonkey data scripts appear similar to HTML, and this is by design. In order to write a data script, the user can view the source of the page that they wish to print, in this example, we will look at a recipe from Foodsville as shown in Figure 6. The idea is that the user can actually scrape data from the page that they see. They do this by copying the pertinent pieces of HTML from the source to their PrintMonkey script. If there are pieces of HTML that they do not care about, they simply replace them with "`...`". The top script in Figure 7 shows an example data script for the Foodsville site. This data script performs string matching so when we see the

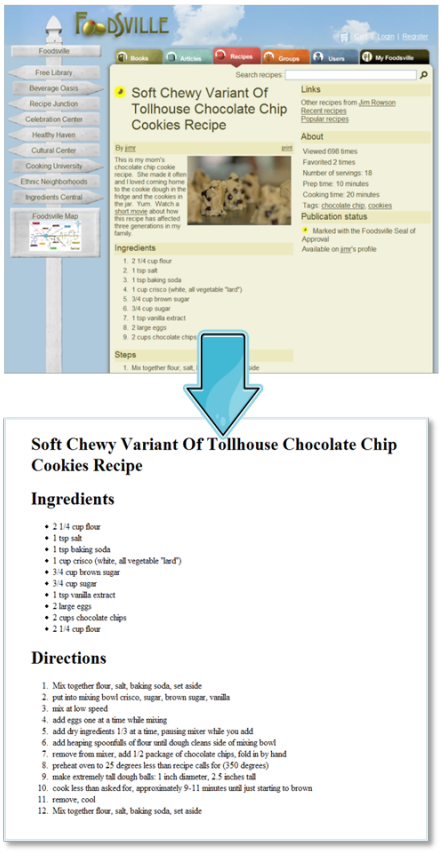

Figure 6: Output from PrintMonkey Scripts

```
<html>
     ...
     <td>
          <h1 class="welcome">$title$</h1>
     </td>
     ...
     $$for each $ingredients[$ingredient]
          <li class="ingredient">$ingredient$</li>
     $$end
     ...
     $$for each $steps[$step]
          <li class="step">$step$</li>
     $$end
     ...
<html>
```

```
<html>
    <head></head>
    <body>
        <h1>$title$</h1>
        <h1>Ingredients</h1>
        <ul>
            $$for each $ingredient in $ingredients
                <li>$ingredient$</li>
            $$end
        </ul>

        <h1>Directions</h1>
        <ol>
            $$for each $step in $steps
                <li>$step$</li>
            $$end
        </ol>
    </body>
</html>
```

Figure 7: Data Mining and Visual PrintMonkey Scripts

line `<h1 class="welcome">$title$</h1>`, this will look for the first part of the string, before `$title$` and the last part after `$title$` and assign whatever is matched to a variable of the name `title`.

We also provide additional functionality such as creating an array. For this, the user can use the `$$for each` construct. The line `$$for each $ingredients[$ingredient]` means that we create an array with the name `ingredients` which will comprise each `ingredient` matched within the `for` loop.

The JavaScript which is generated by this data script is shown in Figure 8. This is generated with the actual HTML of the page so that it is already populated when it is paired with the generated visual script.

There are times when the basic PrintMonkey script we have shown above is not quite powerful enough. Therefore we have provided two additional mechanisms for altering data before we store it. The first is shown in Figure 9. This is an example of a data script which works for Facebook images. In order to convert the small image we see in a Facebook album, to its large counterpart, we need to replace the `/s` in the image URL to a `/n`. This script shows that we can match pieces of a string and then piece it back together before we put it in the image array.

The second approach is by using JavaScript inline in the PrintMonkey data script, as shown in Figure 10. The function is denoted by angled brackets allowing the developer to use JavaScript to alter the data how they see fit. This performs the same function as the previous example with piece matching, but here the developer has more control.

3.3.2 Visual Scripts

Finally, the last type of script that a developer can create is the PrintMonkey visual script. This is quite similar to HTML because this script actually outputs the HTML that you write.

Figure 6 shows a plain printed recipe. The visual script for this page is shown in the bottom of Figure 7. This script is transformed to JavaScript as shown in Figure 11. Unlike the data scripts which contain the actual values, the visual script contains references to those values. We see that each HTML string is put in a `document.write` and each variable is put in as the variable name with + operators. The for loops are constructed for the lists and the index variables are named as `indexjenb0`. When nested, these variable names will become `indexjenb1`, `indexjenb2` and so on.

As with the data scripts, the simple script shown here may not always be solely adequate so we provide additional mechanisms such as:

- break(..): breaks an array up into an array of smaller arrays
- $$pagebreak: creates a page break using `<div style=page-break-after:always;></div>`
- $$position: the position in a for loop
- $$if odd: whether that position is odd
- $$if even: whether that position is even

The break(..) function can be used to break an array up. Revisiting our FaceBook example, if instead of printing full size images as

```
var title = "Soft Chewy Variant of Tollhouse Chocolate Chip Cookies Recipe";

var ingredients = [’2 1/4 cup flour’, ’1 tsp salt’, ’1 tsp baking soda’, ’1 cup crisco (white, all vegetable "lard")’,
’3/4 cup brown sugar’, ’3/4 cup sugar’, ’1 tsp vanilla extract’, ’2 large eggs’, ’2 cups chocolate chips’,
’2 1/4 cup flour’];

var steps = [’Mix together flour, salt, baking soda, set aside’, ’put into mixing bowl crisco, sugar, brown sugar,
vanilla’, ’mix at low speed’, ’add eggs one at a time while mixing’,’add dry ingredients 1/3 at a time, pausing mixer
while you add’, ’add heaping spoonfuls of flour until dough cleans side of mixing bowl’, ’remove from mixer, add 1/2
package of chocolate chips, fold in by hand’, ’preheat oven to 25 degrees less than recipe calls for (350 degrees)’,
’make extremely tall dough balls: 1 inch diameter, 2.5 inches tall’, ’cook less than asked for, approximately 9-11
minutes until just starting to brown’, ’remove, cool’];
```

Figure 8: Generated JavaScript for Data Script

```
<html>
     ...
     <div id="album">
          $$for each $images[$image]
               <img src="$p1$/s$p2$" Ě />
               $image = $p1$ + "/n" + $p2$;
          $$end
     </div>
     ...
</html>
```

Figure 9: Facebook Album PrintMonkey Data Script

shown in Figure 4, the user wants to print 6 images per page and 2 per line, in Figure 12. In order to do this without helper functions, the user could have to keep track of where they are in the array using modulus arithmetic to decide whether they were at a multiple of 2 or 6. Using the break function, they can write this much more simply as shown in Figure 13. This shows how the user can split the images up into arrays of size 6, and assign them each to an array called `page`. Then they can do the same thing for pairs.

3.4 Submitting the Script

An online application for managing scripts is available on the PrintMonkey website. Here a user may sign in using Open-ID [13, 25] and edit any of their existing scripts and also submit new scripts. For each script they submit, along with the template code, they will also have to submit certain information about the script. These items include:

- cue: a word to match the URL of the page, `|` signifies "or" (e.g. `recipe` or `photo|picasaweb|album`)
- title match: regular expression matching the page's title (.* for all)
- include: regular expression matching the URL (.* for all)
- exclude: regular expression for URLs not to match (can be left blank)
- tooltip: tooltip that shows up on the preview page for printing
- include source: for Greasemonkey style scripts only, will include this URL in the head of the printed page

These items are used to decide which scripts match the given page. In addition, each template will have an associated popularity that increases every time the template is selected. By default, if the author is signed in, the pages they have created will show up first followed by those with the highest popularity.

```
<html>
   ...
   <div id="album">
      $$for each $images<<function($image)
         tokenArray = image.split("/");
         fileName = tokenArray[tokenArray.length - 1];
         fileName = "n" + fileName.substr(1);
         newName = "";

         for(var j = 0; j < tokenArray.length - 1; j++) {
            item = tokenArray[j];
            newName += item + "/";
         }

         newName += fileName;
         return newName;
      >>
         <img src="$image$" />
      $$end
   </div>
   ...
</html>
```

Figure 10: Facebook PrintMonkey Data Script with Function

3.5 Debugging

Since the template code is converted into JavaScript, the errors will be JavaScript errors and a Firefox plugin is the best way to debug them. We recommend Firebug [3] in order to see what the actual errors are. Unfortunately due to issues with using multiple JavaScript files, Firebug cannot display the actual JavaScript that caused the error to occur, but only the actual line. Therefore if an error occurs within the generated JavaScript files for PrintMonkey templates, the generated JavaScript is shown in an alert box. This is an easy way to see if the developer has mistyped a variable name or if something is being generated incorrectly. For Greasemonkey and JavaScript templates, nothing is displayed since the error was not caused by our generated code and Firebug alone must be used.

4. EVALUATION

Table 1 shows PrintMonkey in comparison to the other approaches discussed in the Section 1.1. We look at three general areas for evaluation: functionality, customizability and scalability. In functionality, we look at whether elements of the page can be moved or deleted, whether templates can be reused and shared and also whether or not the tool runs primarily on the client as opposed to on the server, which is more costly. For customizability, whether or

Table 1: Comparison of Existing Approaches

		Internet Explorer 7	HP Smart Web Printing	Cascading Style Sheets	Tabblo Print Toolkit	Print Monkey
Functionality	Remove/Move Items		x	x	x	x
	Reusable			x	x	x
	Sharable			x		x
	Client Side	x	x	x		x
Customizability	Users Can Create Layout	x	x			x
	No Programming Knowledge to Create Layout	x	x			x
Scalability	Multiple Sites		x	x		
	Fine Grained Layout		x	x	x	x
	Unmodified Websites	x	x			x
	Interoperable	x	x	x		x

```
document.write("<html>");
document.write("        <head></head>");
document.write("        <body>");
document.write("                <h1>" + title + "</h1>");
document.write("                <h1>Ingredients</h1>");
document.write("                <ul>");
for(var indexjenb0 = 0;
    indexjenb0 < ingredients.length;
        indexjenb0++) {

    document.write("<li>"
        + ingredients[indexjenb0] + "</li>");
}
document.write("                </ul>");
document.write("                <h1>Directions</h1>");
document.write("                <ol>");
for(var indexjenb0 = 0;
    indexjenb0 < steps.length;
        indexjenb0++) {

    document.write("<li>"
        + steps[indexjenb0] + "</li>");
}
document.write("                </ol>");
document.write("        </body>");
document.write("</html>");
```

Figure 11: Generated JavaScript for Visual Script

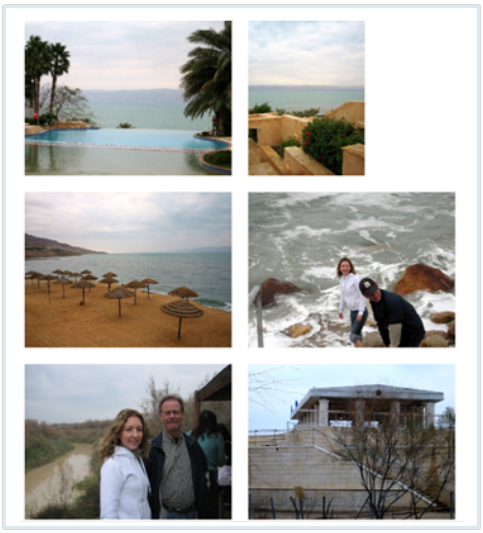

Figure 12: Standard Size Album Image Print

not the users can create their own layouts is important in addition to whether or not they need programming knowledge in order to write those templates. Finally, for scalability, we look at whether the tool can incoporate content from more than one website, whether the layout of elements can be finely controlled, whether a website owner has to edit their website in order to use the tool and whether this tool could be used in conjunction with others in the evaluation.

Internet Explorer or browser printing support does not allow the user much control. They cannot move or delete items, and cannot create, reuse or share templates. However, it does run solely on the client, requires no programming experience from the user and the original site does not need to be modified. Additionally, the user cannot use content from more than one site but printing options can be used with CSS and PrintMonkey as well, since they are also printed from the browser.

HP Smart Web Printing provides users with a way to visually collate data (including moving/deleting elements) to be printed from multiple sites. However, users can only print once and the efforts are not reusable and cannot be shared. We believe this functionality is particularly key in terms of the user friendliness of a web printing tool. This tool runs solely on the client side and is also for users with no programming experience. The original site does not need to be changed and the results of other printing tools in this list could be fed into HP Smart Web Printing as well.

Cascading style sheets can be used by a website owner to ensure their page is printed well. It does so in a way that is completely transparent to the user, and runs within the browser on the client, and could include multiple webpages, since CSS can include additional content to be printed, as well as move and remove elements. However, the user still has no control over the finished product, since they cannot create their own templates, and has no choice over multiple layouts. Since CSS runs within the browser, it can also be used in conjunction with other tools on this list.

Tabblo Print Toolkit is useful when website owners are willing to modify their site to provide TPT with the data it needs for printing. It provides absolute control since it generates PDF and these templates are reusable since the actual website is modified. TPT also provides the user with multiple options for printing a single page, as shown in Figure 14. Figure 15 shows how a page must be modified with CSS selectors in order to select printable data. TPT consumes considerable server resources to generate PDF, however, and one of the goals of PrintMonkey was that it would run primarily on the client. In addition, users cannot create their own templates and TPT cannot print data from multiple sites.

In conclusion, PrintMonkey does not necessarily provide more

```
function __TABBLO_TPT_LOAD() {
    Tabblo.embedded.sites.SettingsObject.preprocess.apply({
        // content definition:
        Properties:
        {
            template: 'recipe'
        },
        Content:
        {
            'pagetitle': { match: 'css', selector: 'span.recipename' },
            'author': { match: 'css', selector: 'span.authorname' },
            'ingredients':     { match: 'css', selector: 'li.ingredient' },
            'instructions':     { match: 'css', selector: 'li.step', outputTagToo: true }
        },
        FixedContent:
        {
            accentcolor2: '#D6CD6E',
            accentcolor: '#6699BD',
            logo: 'http://www.foodsville.com/foodsville/images/logo.jpg'
        }
    }, []);
}
```

Figure 15: CSS Selectors for TPT

```
<html>
<head></head>
<body>
$$for each $page in $images.break(6)
  <table>
  $$for each $pair in $page.break(2)
  <tr>
    $$for each $image in $pair
          <td>
              <div style=height:300px;width:400px;>
                   <img src=$image$ height=\"100%\" />
              </div>
          </td>
    $$end
  </tr>
  $$end
  </table>
  $$pagebreak
$$end
</body>
</html>
```

Figure 13: PrintMonkey Visual Script for Images

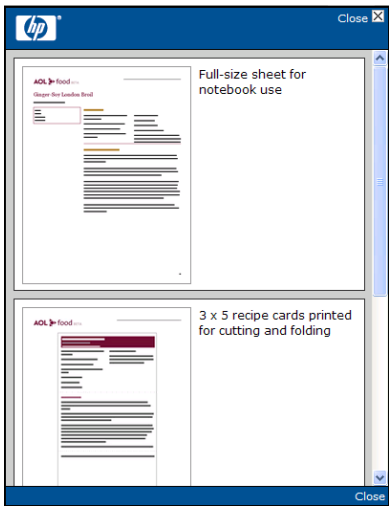

Figure 14: TPT Print Options

features than other approaches. PrintMonkey can be used in situations where a website owner is unwilling to modify their site, the user wants to have multiple printing options and does not want to print data from multiple websites. It is also ideal for those with no programming knowledge to use, especially with sharing capability. It is also easy for more experienced developers to create their own templates in multiple ways. PrintMonkey can also be used in conjunction with other solutions discussed above, since it creates a brand new webpage. However, if a user wants to generate PDF, print data from multiple pages, or does not care about reusability, then another approach may be more suitable.

5. FUTURE WORK

There are various areas of further development for PrintMonkey. These include being able to process dynamic types of information, and being able to process more than one page at once. We would also like to be able to do more with the data we have retrieved and make it even easier for people to create templates. These are all discussed in the following subsections. In addition, it's important to note that future work should also look into copyright issues for printing websites without the authors' permission. For example, some sites might be upset if their advertisements were removed before printing.

5.1 Dynamic HTML

When a user views the source of a GMail [6] message, they cannot see the HTML source of that email. Much of the content on pages visited is dynamically generated. In order to extract information from this content, we will need to be able to grab this dynamic content as it is retrieved. This data may not be HTML and may be something like JavaScript Object Notation (JSON) [12] instead. So we will have to make allowances for that as well.

5.2 Mashups

Currently, PrintMonkey can only print one webpage at a time. Sometimes users may want to print elements from multiple pages

on one sheet of paper. For example, to create personalized travel itineraries. Due to the design of PrintMonkey and having to use Greasemonkey to fetch the contents of the page, we would have to provide additional mechanisms to fetch the HTML of multiple pages at once.

5.3 Data Object Manipulation

There may be times when users need additional information about a data object that the data template does not provide. For example, each image from Facebook currently only has an image URL associated with it. However, a user may wish to also know the image's width and height in order to do automatic positioning with the images, an active area of research [18]. An example of this automatic positioning is shown in Figure 16.

Figure 16: Automatic Layout of Images

5.4 WYSIWIG Template Creation

Finally, there is no way for a user to visually create a print template by designing the output itself. Platypus [14] is a Firefox Plugin that inserts a toolbar into Firefox which allows the user to alter how the page appears within the browser itself. They can perform tasks such as change background, font color, font size, delete items, move items etc. Once the user is finished, they can then save these results to a Greasemonkey script. Currently this plugin could be used to create scripts to upload to PrintMonkey. However this plugin was not created with printing in mind so we would like to have our own tool to create our own PrintMonkey templates. Platypus is shown in Figure 17, as well as the altered view in a browser. The overlay image is of the original page the user is modifying. Platypus highlights the areas that the user is modifying, as shown in the top righthand portion of Figure 17.

6. CONCLUSION

In this paper we have presented the design and prototype implementation of a web printing tool, PrintMonkey. The goal of this tool is to allow users to easily select from multiple ways to print a webpage, without requiring the original webpage to be modified. It also allows users to easily customize how their webpages are printed by writing their own templates that can be used in conjunction with other tools. Developers are given the flexibility to write templates in JavaScript if they are so inclined, or take advantage or our data mining scripts paired with our visual template scripts. Our analysis shows the ways in which PrintMonkey offers improvements over Internet Explorer 7, HP Smart Web Printing, CSS, and TPT in terms of functionality, customizability and scalability. The prototype has provided another point on the spectrum of

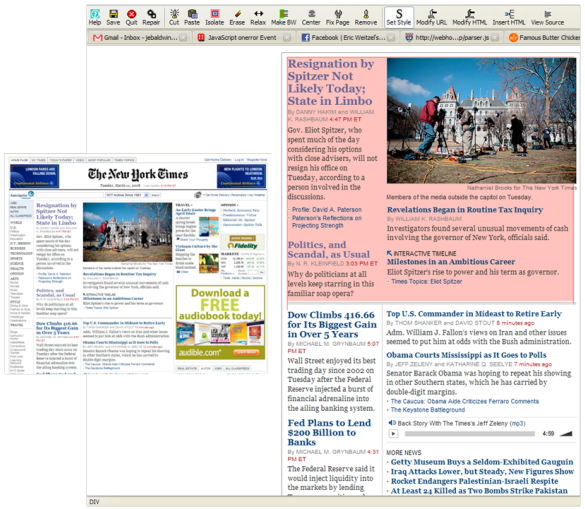

Figure 17: Platypus Firefox Plugin

printing tools, one that we believe will give users maximum benefit for minimum costs.

7. REFERENCES

[1] All Recipes. http://www.allrecipes.com

[2] Cascading Style Sheets. http://www.w3.org/Style/CSS/

[3] Firebug. http://www.getfirebug.com/

[4] Firefox web browser. http://www.mozilla.com/en-US/firefox/

[5] Foodsville. http://www.foodsville.com

[6] GMail. http://www.gmail.com

[7] Greasespot. http://www.greasespot.net

[8] HP Smart Web Printing. http://h71036.www7.hp.com/hho/cache/482779-0-0-225-121.html

[9] IEBlog: Make printing work better with the web. http://blogs.msdn.com/ie/archive/2005/07/31/445778.aspx

[10] Internet Explorer: Home page. http://www.microsoft.com/windows/products/winfamily/ie/default.mspx

[11] JavaScript. www.javascript.com

[12] JSON. http://www.json.org

[13] Open-ID. http://www.openid.com

[14] Platypus. http://platypus.mozdev.org/

[15] Proxomitron.info... the webhiker's guide to proxomitron. http://www.proxomitron.info/

[16] Tabblo Print Toolkit. http://developer.tabblo.com/index.php/tabblo-print-toolkit/

[17] A. Artail and M. Raydan, "Device-aware desktop web page transformation for rendering on handhelds," *Personal Ubiquitous Comput.*, vol. 9, no. 6, pages 368–380, 2005.

[18] C. B. Atkins, X. Lin, and M. I. Enachescu, *Constraint based albuming of graphic elements*. US patent application filed June 10, 2005, 2005.

[19] M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller, "Automation and customization of rendered web pages," in *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 163–172. New York, NY, USA: ACM, 2005.

[20] K. Henricksen and J. Indulska, "Adapting the web interface:

An adaptive web browser," 2001. citeseer.ist.psu.edu/henricksen01adapting.html

[21] G. A. D. Lucca, A. R. Fasolino, M. Mastoianni, and P. Tramontana, "Identifying cross site scripting vulnerabilities in web applications," in *Sixth IEEE International Workshop on Web Site Evolution (WSE'04)*, pages 71–80, 2004.

[22] N. McFarlane, "Fixing web sites with Greasemonkey," *Linux Journal*, vol. 2005, no. 138, p. 1, 2005.

[23] M. K. Qiu, K. Zhang, and M. Huang, "An empirical study of web interface design on small display devices," in *WI '04: Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 29–35. Washington, DC, USA: IEEE Computer Society, 2004.

[24] M. Qiu, K. Zhang, and M. Huang, "Usability in mobile interface browsing," *Web Intelli. and Agent Sys.*, vol. 4, no. 1, pages 43–59, 2006.

[25] D. Recordon and D. Reed, "OpenID 2.0: a platform for user-centric identity management," in *DIM '06: Proceedings of the second ACM workshop on Digital identity management*, pages 11–16. New York, NY, USA: ACM, 2006.