Nyx: A Display Energy Optimizer for Mobile Web Apps

Ding Li, Angelica Huyen Tran, and William G. J. Halfond
University of Southern California
Los Angeles, California, USA
{dingli, tranac, halfond}@usc.edu

ABSTRACT

Energy is a critical resource for current mobile devices. In a smartphone, display is one of the most energy consuming components. Modern smartphones often use OLED screens, which consume much more energy when displaying light colors than displaying dark colors. In our previous study, we proposed a technique to reduce display energy of mobile web apps by changing the color scheme automatically. With this approach, we achieved a 40% reduction in display power consumption and 97% user acceptance of the new color scheme. In this tool paper, we describe Nyx, which implements our approach. Nyx is implemented as a self-contained executable file with which users can optimize energy consumption of their web apps with a simple command.

Categories and Subject Descriptors

D.2.5 [**Testing and Debugging**]: Diagnostics

General Terms

Performance

Keywords

Energy, display, OLED, mobile

1. INTRODUCTION

Many modern smartphones use OLED screens [11], such as the Samsung Galaxy, Sony Xperia, and LG Optimus series. Compared to other types of screens, OLED screens have the unique characteristic that they consume more energy displaying light colors, such as white, than dark colors, such as black. However, many modern mobile web apps are designed with light-colored backgrounds. This means that many web apps are not energy efficient for mobile devices.

To address this problem and improve the energy efficiency of mobile web apps, we proposed a new technique [9] to automatically transform the color scheme of a mobile web application. In our approach, we rewrote the server side code, static pages, and images of a web application so that the resulting web application could generate pages that were more energy efficient when displayed on a smartphone. The rewritten web application can then be made available to OLED smartphone users via automatic redirection or a user-clickable link.

Our approach employs program analysis to model the output of a web app and the potential visual relationships among the colors in the pages and images. Then our approach generates a new color scheme that reduces the energy consumption but still keeps the relative relationship between colors. In our evaluation, we found that the transformed web app can reduce energy consumption by 40% with 97% of users indicating they would find the new color scheme acceptable if the battery power was low.

In this tool track paper, we describe the implementation of Nyx. To use Nyx, developers only need to run the executable file from the Nyx package and it will automatically generate the energy efficient version of the target web app. Then, the developers only need to deploy the energy efficient version of their web app.

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3675-8/15/08...$15.00
http://dx.doi.org/10.1145/2786805.2803190

2. OUR TECHNIQUE

Nyx takes the target web app as the input and outputs the web app with a transformed color scheme. Our technique has three major phases to transform the color scheme of web apps: HTML Output Analysis, Color Transformation, and Output Modification. The working process of Nyx is shown in Figure 1. We only summarize these three phases briefly in this tool paper.

2.1 HTML Output Analysis

The first phase it to model the structure of the HTML files generated by the input web app. This model is called the HTML Adjacency Relationship Graph (HARG) and it shows the visual relationships of HTML tags. The HARG presents similar information to the Document Object Model (DOM). However, the HARG also contains relationships that could be derived from loop generated HTML elements. The HARG can be generated from both the source code and the static HTML file pages.

To generate the HARG from the source code, Nyx first builds another model, the HTML Output Graph (HOG), which describes the HTML pages that can be generated by the web application. Intuitively, the HOG is a projection of the web application's control flow graph (CFG) where all of the nodes are instructions that generate HTML output.

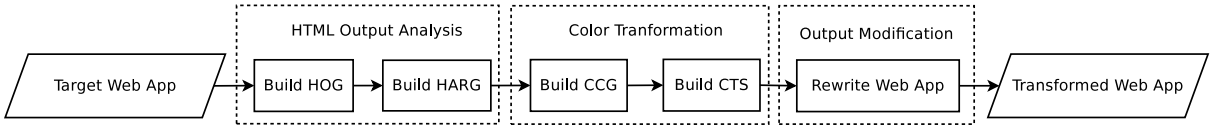

Figure 1: the process of Nyx

To build the HARG, Nyx parses the HOG to find all of the HTML tags in the HOG. Then, Nyx creates nodes in the HARG with each opening HTML tags or self-closing tags. Finally, Nyx connects any two nodes in the HARG if there is a path between them in the CFG.

To generate the HARG for static HTML pages. Nyx parses the static HTML files and builds their DOM, which will be used as the HARG.

2.2 Color Transformation

In the second phase, Nyx generates the new color scheme, which is represented as the Color Transformation Scheme (CTS). The generated CTS will use energy efficient colors and, at the same time, maintain the color relationships between neighboring HTML elements. With the CTS, the newly generated color scheme can reduce display energy consumption and maintain, to some extent, the aesthetics of the web app.

To generate the CTS, our approach first builds a Color Conflict Graph (CCG), which describes the color relationships between pairs of HTML elements that have a visual relationship in the HARG. The CCG is a weighted complete undirected graph defined as $\langle V, v_0, W \rangle$. The set V represents the graph's nodes, where each node represents a color in the HTML page. $v_0 \in V$ is the root-node of the graph, which is the color that will be transformed to black, this color is specified by users. W is a weighting function $W : V \times V \rightarrow I$. Since the CCG is a complete graph, there is an integer edge weight defined for every pair of tuples in V. The weighting function is used to give priority to certain types of visual relationships. Our approach operates on three different types of CCG, the Background Color Conflict Graph (BCCG), which models the relationship between the background colors of neighboring HTML elements; the Text Color Conflict Graph (TCCG), which models the relationship between text colors and their corresponding background HTML element colors; and the Image Color Conflict Graph (ICCG), which models the relationship between an image and its enclosing HTML tag. The BCCG, TCCG, and ICCG vary in the weights attached to certain color relationships.

To transform the colors, our approach first changes the background color of the root node of the CCG to black. Then the approach calculates new colors of other nodes in the CCG so that the color distances between adjacent nodes in the recolored graph are as similar as possible to color distances in the original graph. Because the recoloring problem is NP, we designed a Simulated Annealing Algorithm based technique to approximate an optimal solution.

2.3 Output Modification

In the third phase, the approach rewrites the web application so that it generates web pages based on the CTS. For our approach we have two different mechanisms to do this. For CSS files and HTML templates, our approach simply uses regular expressions to replace all color strings with their new energy efficient colors. In practice we have found that more sophisticated approaches, such as using CSS parsers to identify style properties is unnecessary. For colors that are defined by dynamically generated HTML, the approach inserts instrumentation to perform the rewrite at runtime. The instrumentation replaces the APIs that print HTML to clients (e.g., `JspWriter.println`) with calls to customized printing functions. These printing functions scan the output as it is generated and replace printed colors with their corresponding colors in the CTS.

3. USAGE SCENARIO

There are two typical scenarios in which Nyx could be used to help web app developers build energy efficient web apps. In this section, we discuss how Nyx could be used in these two scenarios.

3.1 Optimizing Existing Web Apps

Owners of a web application may want to make their existing web app energy efficient for mobile devices. These applications may have been developed a long time ago or outsourced to external development teams. It may be expensive or very difficult for the current owner of the web app to rebuild or modify the existing web app.

In this case, Nyx is an effective solution for these web app owners. Nyx can automatically generate an energy efficient alternative web app from the existing version so that owners of the web app do not need to find a technical team to modify the existing version or rebuild it by themselves. The only effort on the side of the web app owners is to deploy the transformed web app and provide an access point for users to visit the transformed web app.

3.2 Providing Guidance to Developers

During the development of a web app, developers may also want to create an energy efficient version of their web app. However, developers do not have enough information about how to design the color scheme of their web apps to make it energy efficient. Providing some preliminary information could reduce the effort required to redesign the color scheme of web apps.

In this case, developers could use Nyx to provide the first, preliminary design of the energy efficient version of their web app. After that, the developers would only need to check the color scheme generated by Nyx and do some refining to improve the aesthetics of the web app.

4. TOOL DESCRIPTION

We implemented Nyx as a Java application. It is a standalone, self-contained jar package that users can use on different platforms, such as Windows, Linux, and Mac OS, that have a Java Virtual Machine (JVM). At this time, Nyx is implemented only for Java or JSP based web applications. In the future, we will extend it to other types of web apps, such as PHP based web apps.

Nyx accepts two inputs: the target web app and its Meta Information File (MIF). The MIF is an XML file that includes five types of information about the web app: the path to class files, the path to design templates, the path to CSS files, the path to all images, the entrance method or associated template file of each web page, and the background color of each web page. The output of Nyx is the optimized version of the web app. Figure 2 shows a comparison of the output of Nyx to the original web app, where the left hand side is the original version and the right hand side is the transformed version produced by Nyx.

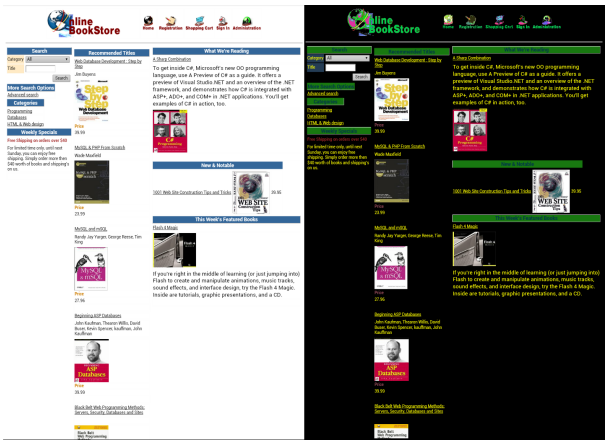

Figure 2: An Example of the output of Nyx

In the HTML Output Analysis phase, we use string analysis and HTML analysis techniques to model the potential output of a web app. The algorithm we used to model string values is based on the method of Yu and colleagues [14]. The input of the String Analyzer is the executable classes of the web app and its output is the HOG, the String Analyzer models the output of the web app and generates the models of potential HTML output. The String Analyzer leverages Soot [4] to build the underlying call graphs, control flow graphs and the Jimple representation of Java classes. During this process, it detects the instructions that print output in Java web apps, such as *JspWriter.println*, and creates a projection of the control flow graph of the web app. After that, the String Analyzer models the output of each print instruction as a Finite State Automaton (FSA). The library we used to manipulate FSAs is the BRICS automaton library [2].

After the String Analyzer is the HTML Analyzer, which analyzes the HOG generated by the String Analyzer and generates the description about the HTML structure, which is the HARG. During this process, the HTML Analyzer also needs to identify the colors of each HTML tag. For colors defined directly in the HTML file, the HTML Analyzer parses the HOG directly. For colors defined in the CSS files, the HTML Analyzer uses the SAC CSS parser [1] to identify colors from CSS files.

In the Color Transformation phase, the CCG Builder builds the CCG to model the color relationships in HTML files. The input of the CCG builder is the HARG generated by the HTML Analyzer and the color palette used by the images, which can be identified by a third party image analyzer, such as CSS drive [3].

After the CCG Builder is the Color Transformer which accepts the CCG as the input and generates the CTS to describe how to change colors. In the Color Transformer, we mapped the color transformation problem to the Energy Minimization Problem[1] (EMP), which is a well-known pixel recoloring problem in the computer vision field [12]. We use a Simulated Annealing Algorithm (SAA) to solve this problem.

In the Output Modification phase, the App Generator takes the CTS as input and generates the optimized version of the web app. For the App Generator, we used the BCEL library to modify Java classes and Perl script to modify colors in the CSS files. Our implementation handles HTML 4 and CSS 2, but it is straightforward to extend our tool to support HTML 5 and CSS 3.

5. TOOL USAGE

Nyx is packaged as a self-contained executable jar. To run Nyx, users only need to run the executable jar package and provide the input, which is the path to the app and its MIF. During its execution, Nyx generates intermediate artifacts after each step.

During the phase of HTML Output Analysis, Nyx first generates the HOG as a dotty file, which contains the information about the generated HTML. The HOG of the web app shown in Figure 2 is visualized in Figure 3. Its contents, which are rendered here, are basically a skeleton of the HTML of the web app that does not contain the images and contents, only the tags.

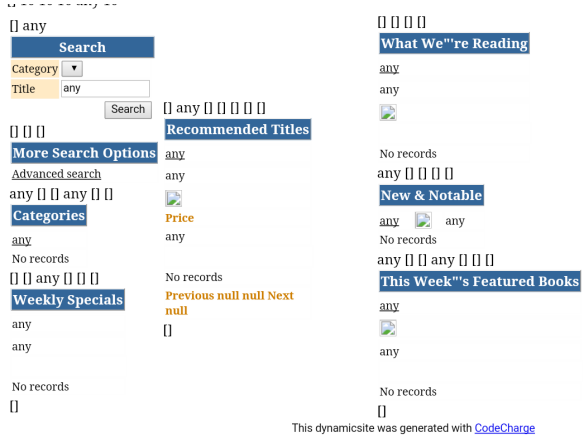

Figure 3: The HOG of the web app

Based on the HOG, Nyx generates the HARG, which describes the relationship of HTML tags in the web app. Then Nyx generates the CCG to describe the color relationships. One example of the CCG of the web app in Figure 2 is shown in Figure 4. This is a BCCG, which describes the relationship of background colors in the web app. Each node in the CCG has the node ID and the RGB value of its color. In this example, there are three background colors: white (RGB: 255,255,255), royal blue (RGB: 51,102,153), and light tan (RGB: 255,234,197). The numbers on the CCG edges show the relationship between each pair of colors. The value 3.0 indicates that there is a parent-children relationship between the tags and 0.5 indicates that the two colors are not related to each other. In our example, white is the color of the body

[1]Here, the term "energy" refers to general cost

tag of the HTML file, which contains the tags with royal blue and tags with light tan, so the edges between white and the other two colors have a relatively high weight. The tags of royal blue do not have any sibling or parent-children relationships, so the edges between them have a much smaller weight. These weights will be used to prioritize which color relationships should be given higher precedence to maintain.

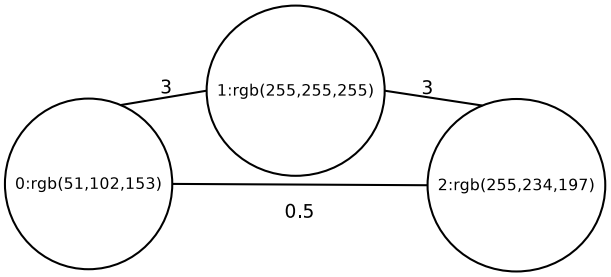

Figure 4: The CCG of the web app

The CTS is calculated based on the CCG, it is implemented as a set of generated scripts. For Java executable files, the CTS is Java code that can instrument the Java classes and replace colors defined in the classes. For CSS files, the CTS is a Perl script that can replace color strings. For images, the CTS is an imagemagic script that can replace colors in images.

6. EVALUATION

We performed an empirical evaluation of Nyx. To evaluate Nyx, We used seven open source Java-based web applications to evaluate our approach, including applications that have been used in related work, to ensure a broad representation of implementation styles. These applications are implemented using different web application frameworks, have colors defined by HTML and CSS, and employ a varying amount of JavaScript in their user interfaces.

In our evaluation, on average, it took less than 2 minutes for Nyx to optimize a web app. The transformation itself does not introduce any significant overhead to the runtime of each web app

In our experiments, there was a 25% decrease in energy consumption during the Loading and Rendering phase and 40% less power consumed during the Display phase for the transformed web applications. We conducted an empirical study on user acceptance of Nyx and found that more than 97% of users would choose to switch to the transformed version before the battery became critically low.

7. RELATED WORK

The closest work to Nyx is Mian and colleagues' work Chameleon[6]. However, this approach is a manual approach instead of the automated approach in Nyx. Kamijoh and colleagues' work [8] is one of the first to optimize energy for OLED screens. However, this work only considers two colors, the black background color and the white foreground color.

Choi and colleagues' method [5] and Lyer and colleagues' [7] method save display energy by dimming the screen. Compared with this approach, Nyx can better maintain the readability of the entire page.

Other techniques are also proposed to optimize the energy of mobile devices, for example Pathak and colleagues' work [10]. However, these techniques are not related to display energy.

Another piece of our previous work, dLens [13], detects display energy hot spots in Android apps. However, it is not for energy optimization and cannot generate an energy optimized version of an app.

8. CONCLUSION

We introduce a tool, Nyx, which can automatically transform colors to save display energy of mobile web apps on OLED based devices. The idea of Nyx is to change light colors to dark colors. Nyx itself is implemented as a command line tool that users can simply type the command and generate the energy efficient version of their web apps. In our evaluation, Nyx can achieve 40% savings of display power. At the same time, 97% of users accept the transformed color schemes if the battery goes to critically low.

Acknowledgments

This work was supported by NSF grant CCF-1321141.

9. REFERENCES

[1] `http://cssparser.sourceforge.net/`.
[2] `http://www.brics.dk/automaton`.
[3] `http://www.cssdrive.com/imagepalette/`.
[4] `http://www.sable.mcgill.ca/soot/`.
[5] I. Choi, H. Shim, and N. Chang. Low-power Color TFT LCD Display for Hand-held Embedded Systems. In *ISLPED*, 2002.
[6] M. Dong and L. Zhong. Chameleon: A Color-adaptive Web Browser for Mobile OLED Displays. In *MobiSys*, 2011.
[7] S. Iyer, L. Luo, R. Mayo, and P. Ranganathan. Energy-Adaptive Display System Designs for Future Mobile Environments. In *MobiSys*, 2003.
[8] N. Kamijoh, T. Inoue, C. M. Olsen, M. T. Raghunath, and C. Narayanaswami. Energy Trade-offs in the IBM Wristwatch Computer. In *ISWC*, 2001.
[9] D. Li, A. H. Tran, and W. G. J. Halfond. Making web applications more energy efficient for oled smartphones. In *ICSE*, 2014.
[10] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff. What is Keeping My Phone Awake?: Characterizing and Detecting No-sleep Energy Bugs in Smartphone Apps. In *MobiSys*, 2012.
[11] J. Shinar and V. Savvateev. Introduction to Organic Light-Emitting Devices. In J. Shinar, editor, *Organic Light-Emitting Devices*, pages 1–41. Springer New York, 2004.
[12] O. Veksler. *Efficient Graph-based Energy Minimization Methods in Computer Vision*. PhD thesis, Ithaca, NY, USA, 1999. AAI9939932.
[13] M. Wan, Y. Jin, D. Li, and W. G. J. Halfond. Detecting display energy hotspots in android apps. In *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, April 2015.
[14] F. Yu, T. Bultan, M. Cova, and O. Ibarra. Symbolic String Verification: An Automata-Based Approach. In *Model Checking Software*. Springer Berlin Heidelberg, 2008.