Making Web Applications More Energy Efficient for OLED Smartphones

Ding Li, Angelica Huyen Tran, William G.J. Halfond

University of Southern California

Work supported by NSF Grant CCF-1321141
Motivation

• Energy is a critical resource for smartphones.
• Screen is one of the dominant energy consuming components.
• Smartphone users access web applications.
• Most web applications are not designed to be energy efficient.
Other Techniques

• Dim the display
  – Good start, but more can be done

• Invert colors:

• Chameleon:
  – Requires customized browser
  – Needs additional server infrastructure
  – Color schemes are manually generated
OLED Screens

- Popular technology for smartphone displays
- More energy efficient than prior technologies
- Different energy consumption patterns
Goal

Automatically transform the implementation of a web application so that the web pages it generates consume less energy when displayed on an OLED smartphone.
Challenges

1. Identify the colors used in a web application – Necessary to compute the web pages that can be generated by the application
2. Determine the relationship of colors in the web application – Important ones: adjacency and enclosure
3. Transform colors into a more efficient scheme – Maintain usability and attractiveness (aesthetics)
## Approach Overview

<table>
<thead>
<tr>
<th>Phase 1</th>
<th>Phase 2</th>
<th>Phase 3</th>
</tr>
</thead>
</table>
| 1. Compute the set of generated HTML pages | 2. Determine visual relationships in pages  
- Example: adjacent and contained | 3. Identify colors that have visual relationships |
| 4. Solve for a new color scheme  
- Is more energy efficient  
- Maintains similar color differences | 5. Rewrite application to use new color scheme |
Phase 1: HTML Output Analysis

A. Compute the set of HTML pages that could be generated by the application at runtime

B. Determine visual relationships among HTML elements in the pages
   - Example: adjacent and contained
Phase 1A: HTML Output Graph

- A projection of the CFG, based on work by Moller and Schwarz
- The nodes are the instructions that print content to the web page
  - For example: `JspWriter.println()`
  - The value is represented as FSA
  - FSA generated based on work by Yu and colleagues
- The edges are the paths in CFG that connect each printing instruction
Phase 1B: Visual Relationship Graph
2: Color Transformation

Color Conflict Graph (CCG)

• Shows visual relationships of colors in a page

• BCCG: weights are in \{a, b, c\}
  – a > b > c > 0
  – a: parent-child
  – b: siblings
  – c: everything else
2: Color Transformation

Building the Color Conflict Graph

1. Basic unit is color definition (CD)
   – CSS based
   – HTML based

2. Perform reachability analysis over visual relationship graph

3. “Reaching CDs” define edges in CCG
2: Color Transformation

BCCG: weights are in \{a, b, c\}, \(a > b > c > 0\)
- a: parent-child
- b: siblings
- c: everything else
2: Color Transformation

Generate the color transformation scheme (CTS)

1. Let $S = \langle C_0, C_1, C_2, ..., C_k \rangle$ nodes of the CCG
2. Let $S'$ be the new coloring, where $C_0 =$black
3. Compute $S'$ that results in similar color differences as in $S$, i.e. minimize:

$$\sum_{i=0}^{k} \sum_{j=0}^{k} w_{ij} |\text{Dist}(C_i, C_j) - \text{Dist}(C_i', C_j')|$$

4. Optimization problem is NP-Hard, use simulated annealing to approximate optimal solution
Phase 3: Output Modification

1. Dynamically generated HTML pages
   – Insert instrumentation to replace HTML printing instructions
   – Replace original colors with new colors

2. Template based frameworks
   – Use CSS parser to identify entries to be replaced
   – Replace entries by rewriting CSS and HTML
Evaluation

• **RQ 1:** How much time does it take to generate the new color scheme?
• **RQ 2:** How much energy is saved by the transformed web pages?
• **RQ 3:** To what degree do users accept the appearance of the transformed web pages?
### Subject Applications

<table>
<thead>
<tr>
<th>Name</th>
<th>Framework</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bookstore</td>
<td>JSP</td>
<td>24,305</td>
</tr>
<tr>
<td>Portal</td>
<td>JSP</td>
<td>21,393</td>
</tr>
<tr>
<td>JavaLibrary</td>
<td>JSP &amp; Servlet</td>
<td>73,468</td>
</tr>
<tr>
<td>ClassRoom</td>
<td>JSP</td>
<td>5,127</td>
</tr>
<tr>
<td>Roller</td>
<td>JSP &amp; Struts</td>
<td>154,065</td>
</tr>
<tr>
<td>Scarab</td>
<td>Velocity &amp; Turbine</td>
<td>145,435</td>
</tr>
<tr>
<td>jForum</td>
<td>Velocity</td>
<td>31,841</td>
</tr>
</tbody>
</table>

- Four embed color information in HTML, three use CSS
- Three heavily use JavaScript in the user interface
- Three use Model-View-Controller style
RQ1: Time Cost

- Most of the load time was Soot processing
- Load times varies because some apps use templates
- Transform time varies based on complexity of HTML page structure

All less than three minutes.
RQ2: Energy Savings

Loading Energy decrease: 25%
Display Power decrease: 40%
RQ3: User Acceptance

- Users asked to rate before/after color transformation produced by our approach
- Subject pool: 20 graduate level students
  - 17 responses received
  - Students unaware of project goal
  - No incentives offered
  - Anonymous
1. How do you rate the readability?
2. How do you rate the appearance?
3. If the version on the right could save you X% of the energy, at what battery level would you choose to use it?
   a) Always – regardless of battery level
   b) Most of the time
   c) Only when the battery level is low
   d) Only when the battery level is critical
   e) Never
Attractiveness

Average decrease of 17%

Readability

Average decrease of 15%
• 60% choose transformed app for general usage
• 97% choose transformed app for battery critical
Summary

• **Goal:** make web pages more energy efficient for display on OLED phones

• **Mechanism:** static analysis and transformation of the app implementation

• **Results:**
  1. Analysis takes less than 3min
  2. Energy savings average 40%
  3. 60-97% of users choose transformed version when energy tradeoff is known
The End

Thank you
Approach Implementation

• Built fully automated tool: 
  - Soot: for call graphs and control-flow
  - BRICS Automaton library: for string FSAs
  - SAC CSS Parser: for identifying colors of HTML elements
  - BCEL & Perl: for modifying the web apps
• Implemented for HTML 4 and CSS 2
RQ: Runtime Overhead

- Average overhead of 2.4% (about .168s)
- Standard deviation was 5.6%
- Overhead dominated by transmission variance
- Server-side was higher, about 22%
  - Distribution was bi-modal based on framework