Interaction-based User Interface Redesign*

Luis A. Leiva
ITI – Institut Tecnològic d'Informàtica
Universitat Politècnica de València
llt@acm.org

ABSTRACT

This paper demonstrates a general framework to restyle UI widgets, in order to adapt them to the user behavior. Different implementation examples illustrate its feasibility. The value of this methodology comes from the fact that it is suited to any application language or toolkit supporting structured data hierarchies and style sheets; e.g., interfaces created in HTML, XUL, Flex/AIR (ActionScript), or Java. As described in the paper, an explicit end user intervention is not required, and changes are gradually applied so that they are not intrusive for the user.

Author Keywords

Adaptive Interfaces, Implicit Interaction, Redesign

ACM Classification Keywords

H.5.2 User Interfaces: User-centered design

INTRODUCTION

In computing systems, technology alone cannot survive without adequate user interfaces. To maximize the benefits that usable interfaces bring to users, often developers try to target as many people as possible. However, attempting to create UIs by following the *one-size-fits-all* approach is doomed to fail if an application is intended to be exposed to an arbitrary audience — take for instance web pages or mobile applications. As such, proposals that involve active end user manipulation have been considered to adapt the appearance of UIs (e.g., [1]). Nonetheless, user-driven customization requires to perform additional activities beyond the main purpose of using the application. Claypool et al. [2] observed that every user interaction can contribute to an *implicit interest indicator*; therefore alternative adaptation approaches without burdening the user can be derived. What is more, as stated by Gajos and Weld [3], the rendering of an interface should reflect the needs and usage pattern of individual users. This work is inspired by these ideas.

*An accompanying video submission is available. This work is supported by the Spanish research programme Consolider Ingenio 2010: MIPRCV (CSD2007-00018).

IUI'12, February 14–17, 2012, Lisbon, Portugal.
ACM 978-1-4503-1048-2/12/02.

It is commonly agreed that adaptive systems should accommodate the UI to the user, but also that doing so automatically is a non-trivial problem. Adaptation should be predictable, transparent, and discreet, so that the changes introduced to the UI do not confuse the user. Also, adaptation should not interfere with the structure of the application.

The rationale of this approach follows the Attentive UIs concept [5], i.e., weigh the importance of the information supplied with estimated priorities in user activity. This way, by leveraging information that is submitted with little or no awareness (e.g., mouse movements, clicks, key strokes) those elements (*widgets* from here onwards) where users focus their interaction are incrementally mutated. Specifically, due to the fact that exertions are preceded by attention most of the time, the importance of an interaction towards a specific widget is measured as the proportion of UI-generated events on that widget between consecutive sessions, as shown in Figure 1 and described in Implementation.

SYSTEM OVERVIEW

With the growing popularity of web-based applications, the Cascading Style Sheets (CSS) paradigm has been widely adopted by several programming environments beyond the browser. For instance, it is possible to use CSS in Java[1], GTK+[2], and Qt[3]. CSS allows attaching styles to the application, decoupling the data model and its presentation. This way, in this system, adaptation operates by automatically overriding the rendering of widgets, by simply modifying their CSS. The technique was first introduced in [4], which has been now reformulated to generalize to structured applications (e.g., document object models and scene graphs).

Adaptation Protocol

Initially, the developer indicates which widgets and which properties can be restyled by the system, by using a straightforward JSON notation. Later, when the application is loaded, event listeners will track such widgets in the background. While working with the application, the system "learns" from the user interactions, so that the next time the application is loaded, the visual appearance of the widgets the user has interacted most is subtly modified. Finally, when the user leaves the application, interaction data are serialized and stored on a local database. Figure 2 summarizes the architecture of this framework.

[1] http://weblogs.java.net/blog/2008/07/17/introducing-java-css
[2] http://gnomejournal.org/article/107/styling-gtk-with-css
[3] http://doc.qt.nokia.com/4.3/stylesheet.html

(a) Session #1 **(b)** Session #2

Figure 1: This adaptation technique is tightly coupled to the user interactions, but also delimited by the UI developer. In this example, the system is allowed to modify the `border-width` and `border-color` properties of the search box, and the `padding-top` property of the list items. In the first session (1a) the list is hovered, and hence the padding is increased accordingly. In the second session (1b) the previous redesign is recovered from the local database. Since this time the user does not interacted with the list items, the padding is therefore decreased.

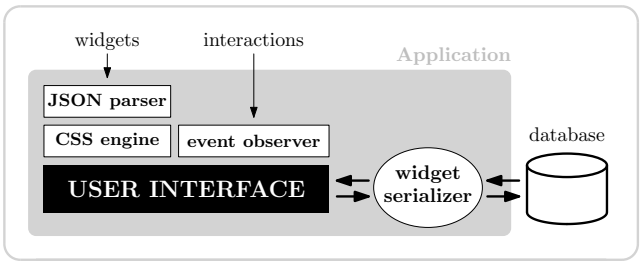

Figure 2: System architecture. Adaptable widgets are indicated by the developer, which will be modified according to how users interact.

Implementation

The API exposes a single method to invoke the system (Figure 3), which takes a configuration object as mandatory argument. Note that such an object notation is dramatically less verbose than the syntax envisioned in [4], and that it can support as many combination of selectors as the underlying CSS engine can implement. This reverts in a wide range of adaptation possibilities, as shown in Figure 1.

```
interface UIAdapt {
  void setWidgets(Object config, Object context);
};
```

```
import org.ui.my.classes.Adapt;
public static void main(String[] args) {
  Adapt.setWidgets({
    "#title": ["font-size", "color"],
    "JFrame > JSlider": ["top", "margin"]
  }, this);
}
```

Figure 3: API definition in Interface Description Language (top) and a conceptual Java Swing implementation example (bottom).

Once the JSON configuration is parsed, object keys are used to select the marked widgets (eg., a button, a table cell, or a text paragraph). Additionally, a series of event lists (hash tables) are created to log interaction data at runtime (e.g., `mousemove`, `click`, or `keydown` event lists). Such an event lists are thus composed of a serialized widget representation as keys and an interaction score as values — the more the user interacts with a widget during a session, the higher the score and vice versa. Scores are not cumulative but differential, following a non-linear distribution (using sigmoid functions) bounded to the interval $]-1.0, 1.0[$ in order to ensure that adaptations will be incrementally applied [4]. Take for instance a button that is allowed to adapt its size by scaling, having a design value of, say, 20px. If the computed score is 0.3, the new size will be $20(1 + 0.3) = 26$px. Conversely, if afterwards the computed score is -0.1, the new font size will be $26(1 - 0.1) = 23.4$px. Event lists are the only user information stored in the local database.

EVALUATION

In terms of system performance, adaptation takes just a few milliseconds to complete (5 ms on average for 50 defined elements with 10 CSS properties each on a traditional computer). Regarding human assessment, an informal study involving 12 users revealed that adaptation works adequately, being also not perceived as distracting most of the time (9 users). These results indicate that this technique supports their intended goals. Nonetheless, I will report on a systematic user evaluation in future papers, to ensure its validity.

DISCUSSION

Building interfaces for each type of device and every kind of user is not scalable for human programmers, therefore, an automated solution is necessary [3]. I have presented an alternative to redesign interface widgets that operates unobtrusively both for the user and the application structure. Substantial improvements can be made at no cost, since the system is the only responsible of performing the adaptation, being delimited by the (implicit) user interactions and the restrictions imposed by the developer (so that not all events affect all styling). A limitation of this framework is that only *numerical* CSS properties can be adapted, i.e., those related to *dimensions* and *colors*. Therefore, more advanced adaptation strategies such as re-arranging several UI components (beyond alignment) or inserting/removing contents would require a technically more sophisticated approach.

REFERENCES

1. Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R. C. Automation and customization of rendered web pages. In *Proc. UIST* (2005), 163–172.
2. Claypool, M., Le, P., Waseda, M., and Brown, D. Implicit interest indicators. In *Proc. IUI* (2001), 33–40.
3. Gajos, K. Z., and Weld, D. S. SUPPLE: Automatically generating user interfaces. In *Proc. IUI* (2004), 93–100.
4. Leiva, L. A. Restyling website design via touch-based interactions. In *Proc. mobileHCI* (2011), 91–94.
5. Vertegaal, R. Attentive user interfaces. *Commun. ACM 46*, 3 (2003), 31–33. Editorial note.