Chapter 5

Adaptive User Interfaces

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. Therefore, we must look for automated solutions.

This chapter proposes a novel approach to automatic UI adaptation that leverages implicit interactions to weight the importance of the information supplied with estimated priorities in user activity. This way, by analyzing information that is submitted with little or no awareness (e.g., mouse movements, clicks, keystrokes), elements where users focus their interaction are incrementally mutated. While this is still a work in progress, preliminary results indicate that this method has an interesting potential to build self-adaptive UIs.

Chapter Outline

5.1 Introduction

In computing systems, technology alone cannot survive without adequate user interfaces. Personalization and customization have been widely promoted by UI design theories but seldom few of them are put into practice. The vast majority of UIs are visually-oriented and assume that users do not have functional impairments of special requirements. Caveats, standards, and best practices have evolved on where to place layout widgets, navigation items, and body content. As such, 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, where, in addition, the range of screen sizes and the rendering possibilities are exceedingly large.

UI adaptation is about exploiting some features of the application and avoiding others. For instance, the mobile space has an incommensurable range of devices, and content often renders better when tailored to specific device characteristics. Another part of adaptation requires working around problems found in specific parts of the UI; e.g., elements that may cause confusion or frustration to first-time users and so on. A more drastic option is to build a separate UI for each user, but a manual approach is impractical and definitely not scalable. Also, continuously performing usability tests to assess new changes committed on the application is very time-consuming. Therefore, we must seek automated adaptation solutions.

Traditionally, UI adaptation techniques can personalize the layout presentation (e.g., modifying font sizes or applying some accessibility guidelines), but unfortunately the changes they perform operate from a global perspective. Some proposals that involve active end user manipulation have been considered; e.g., [Bolin et al., 2005]. Nonetheless, user-driven customization requires to perform additional activities beyond the main purpose of using the application. Some researchers [Arroyo et al., 2006; Atterer et al., 2006; Claypool et al., 2001] have demonstrated that every user interaction can contribute to enhance the utility of the system, therefore alternative adaptation approaches without burdening the user can be derived. What is more, as stated by Gajos and Weld [2004], the rendering of an interface should reflect the needs and usage patterns of their users. This work is inspired by these ideas.

We propose a novel approach that is based on implicit HCI to weight 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, keystrokes), 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 (see Section 1.1.1), the importance of an interaction toward a specific widget

is measured as the proportion of UI-generated events on that widget between consecutive sessions, as described in Section 5.3.

5.2 Related Work

The idea of adapting the UI of applications or even full websites according to user interactions is not new (see, e.g., [Zhang, 2007]). However, practical examples have been too scarce so far. Despite considerable debate, automatic adaptation of UIs remains a contentious area [Gajos et al., 2008]. Commonly cited issues with adaptive interfaces include lack of control, predictability, transparency, privacy, and trust [Findlater and McGrenere, 2008].

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. We believe that adaptation should be both transparent and discreet, so that the changes introduced to the UI do not confuse the user. We also believe that adaptation should not interfere with the internal structure of the application.

Probably the major advances in the field of automatic adaptation of UIs are the ones carried out by Gajos and co-authors [Gajos and Weld, 2004; Gajos et al., 2007, 2008], where adaptation is approached as an optimization problem. However, their experiments were performed on form-based layouts, by modeling widget constraints, and choosing the best alternatives from a defined set of UI elements (e.g., sliders, combo boxes, radio buttons, etc.). Other types of applications such as web pages are nevertheless a completely different matter. Their dynamic nature per se makes the automatic adaptation a challenging task.

On the Web, with the exception of customizing font preferences, browsers do not provide end users with substantial control over how web pages are rendered. This way, researchers have proposed different approaches to layout adaptation that mainly involve user's manual work. Ivory and Hearst [2002] employed learned statistical profiles of award-winning websites to suggest improvements to existing designs; however, changes would be manually implemented. Tsandilas and Schraefel [2003] introduced an adaptive link annotation technique, although it required the user to perform direct manipulation of a middleware application. Notable approaches in this direction include the work of Bila et al. [2007], where the user must actively modify the layout contents. Kurniawan et al. [2006] proposed to override the visual tier of a web page with custom style sheets, but unfortunately updates had to be performed by hand. Now that web standards have minimized browser inconsistencies, this approach can be automatically exploited to automate the adaptation of web design (and other applications, as discussed later) without disrupting users' interaction habits.

(a) Original page **(b)** Automatically adapted design

Figure 5.1: An example of website design modifications. Changed parts are numbered in Figure 5.1b. ❶ headline text: `font-size`, `padding-top`; ❷ navigation menu: `font-size`; ❸ welcome paragraphs: `font-size`; ❹ 'read more' links: `color`; ❺ 'online booking' heading: `color`; ❻ submit button: `font-weight`; and ❼ 'special menu' div: `margin-top`.

5.3 ACE: An Adaptive CSS Engine

Our approach, being based on implicit interaction, allows to gather much usage data without burdening the user. On the other hand, though, collected data are potentially noisy and prone to some errors if not treated adequately. For that reason, the novelty of this approach is two-fold: *1)* to let the webmaster decide *which* elements are going to be adapted; and *2)* to automatically apply slight modifications to the rendering of UI elements based on *how* the user has interacted with them. This way, the system will try to invisibly improve the user-perceived performance toward a UI (Figure 5.1).

The main difference with other state-of-the-art interface adaptation techniques is ours relies on the developer (or webmaster) control to accommodate the appearance of the UI (or page) to the users in a transparent way. This way, our approach aims to focus rather than distract the user.

5.3.1 Rationale

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

[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

the data model and its presentation. This motivated us to develop ACE, an Adaptive CSS Engine in which adaptation operates by automatically overriding the rendering of widgets, by simply modifying their CSS. The technique was first introduced by Leiva [2011], and has been now reformulated to generalize to structured applications (e.g., document object models and scene graphs).

5.3.2 Overview

ACE leverages implicit interactions to incrementally mutate the appearance of interacted widgets (e.g., DOM elements). The importance of an interaction toward a specific widget is measured as the proportion of UI-generated events on that widget between consecutive sessions. Implicit interaction is used thus as a proxy of user attention. The idea is to introduce ephemeral changes that can be easily incorporated and do not alter the UI design in a way that it might confuse the user [Leiva, 2011, 2012a].

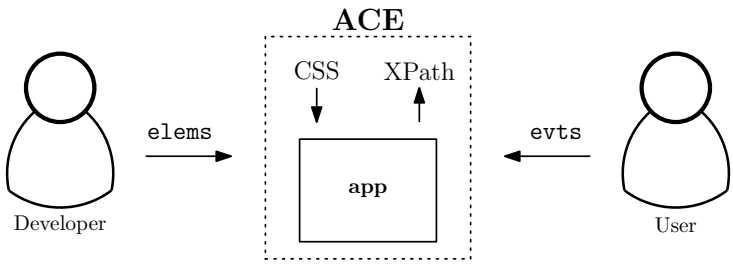


Figure 5.2: Workflow diagram. ACE tracks elements indicated by the developer. When the user access an application, UI events translate interacted elements into XPath notation (or a similar representation) for later storing. On returning to the application, the CSS properties of such stored elements are restyled accorded to computed scores.

ACE was written as a completely self-contained JavaScript (JS) program that restyles *numerical* CSS properties, i.e., those related to:

- **Dimensions** (e.g., `font-size`, `margin-top`). These properties often do have a unit of measure, e.g., `16px`, `2.5em`, or `20%`, which is preserved once they are adapted.
- **Colors** (e.g., `background-color`, `border-color`). These properties do have an hexadecimal representation,which is specified either by a keyword (e.g., `"red"`) or by a numerical RGB specification (e.g., `#RRGGBB` or `rgb(R,G,B)`).

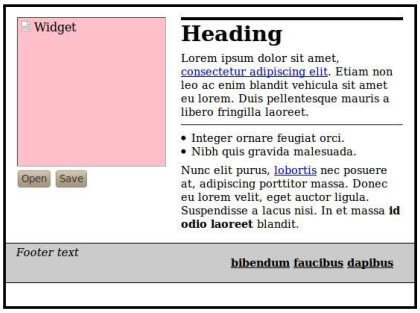
The main features of ACE are summarized in the following list:

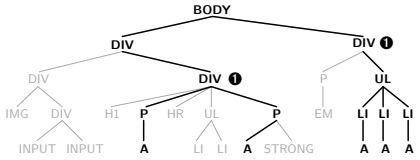
- Does not require end user intervention.
- Supports desktop, touch, and mobile web clients.

(a)

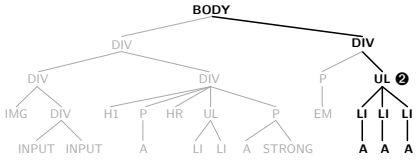
```
ACE.adapt({
  "div a": ["font-size", "color"], ❶
  "p ul" : ["font-weight", "margin"]
});
```

```
ACE.adapt({
  "div + a": ["font-size", "color"],
  "p + ul" : ["font-weight", "margin"] ❷
});
```





(b) pattern: `E F`



(c) pattern: `E + F`

Figure 5.3: Original page design (5.3a) with an overlaid mouse behavior that may cause different adaptation possibilities, according to the following CSS combinator patterns: [5.3b] F elements that are descendants of E elements; [5.3c] F elements immediately preceded by E elements; *Top row*: Sample JSON syntax. *Middle*: Corresponding page changes. *Bottom row*: DOM tree traversals, highlighting in bold the matched paths. Any combination of CSS selectors is supported, e.g., `"div + p.foo > span a:first-child"`.

- Any combination of CSS selectors can be used.
- Modifications are incrementally applied, ensuring that they are not intrusive for the user.
- Adaptation can be performed once the DOM is parsed or the application is fully loaded, so that third party or JS-controlled modifications are also

supported.
- Since the system has a *user interaction history*, it can populate adaptation to other widgets that share a similar structure.

5.3.3 Adaptation Protocol

Initially, the developer indicates which widgets and which properties can be restyled by the system, by means of straightforward JSON notation (see sample code snippets in Figure 5.3). Later, when the application is loaded, event listeners will track such widgets in the background. While using the application, the system "learns" from user interactions, so that the next time the application is loaded, the visual appearance of the widgets the user has interacted most with is subtly modified. Finally, when the user leaves the application, interaction data are serialized and stored into a local database. Figure 5.4 summarizes the architecture of this framework.

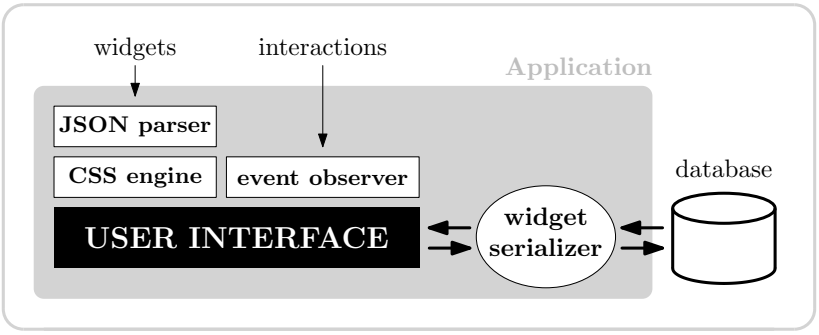


Figure 5.4: System architecture. Adaptable widgets are indicated by the developer, which will be modified according to how users interact with the application.

5.3.4 Implementation

A very simple API was designed to invoke the system. ACE exposes two public methods: `listen()` and `adapt()`. The former allows the developer to prioritize the importance of UI events (e.g., Should a `mousemove` event be assigned lower priority over a `click` event?). The latter takes two arguments (Figure 5.5): a configuration object and a context (the whole application by default).

Under the hood, the elements that were specified in the configuration object as CSS selectors are retrieved by means of the `querySelectorAll()` method or a similar alternative (depending on the programming language). Interaction data are then classified into different event lists, e.g., hovered, typed, scrolled, or tapped elements; where each list member is composed of a serialized widget representation as a key (to allow retrieving them later on subsequent user visits, see bottom rows of Figure 5.3) and an interaction score as a value. The scoring

```
interface ACE {
  void listen(Object eventList, Boolean keepOtherPriorities);
  void adapt(Object config, Object context);
}
```

Figure 5.5: ACE's API definition in Interface Description Language (IDL).

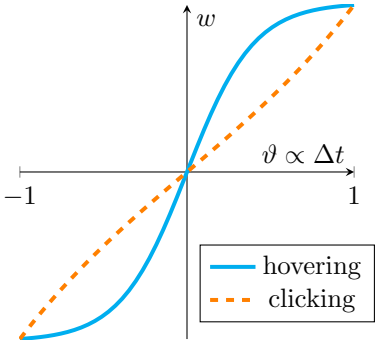


Figure 5.6: Weighting interactions example. Hovering is weighted according to $w = \tanh(\lambda\vartheta)$, while clicking is weighted as $w = \sinh(\lambda\vartheta)$. The parameter λ allows to tune the slope of both curves.

scheme is described in the next section. Basically, a score is proportional to the number of browser-generated events, or, in other words, how many times the user has interacted with UI elements.

Finally, data are persistently stored on the client side by means of an abstraction layer of different storage backends (e.g., `localStorage`, `IndexedDB`, or equivalents), so that the users' privacy is completely under their control; e.g., they may opt to configure their application or browser to restrict access to the storage context, or automatically delete stored data after some time.

5.3.5 Interaction Scoring Scheme

As commented above, each interacted element is assigned a score s, which depends on the event type. For instance, `mousemove` events are triggered in much more quantity than `mousedown` or `keyup` events, and as such they should be weighted accordingly. Let n_i be the number of times an event of type i was fired for a certain widget, and let N be the number of all fired events during application usage. The assigned score for that event is

$$s_i = \zeta(n_i/N) \tag{5.1}$$

where $\zeta(\cdot)$ is a symmetric sigmoid function. The idea is to get scores follow a non-linear distribution, in order to ensure that adaptation is smoothly applied.

Note that if an element receives different types of interactions (e.g., an `input` text field can listen to `click`, `focus`, or `keydown` events) then its scores need to be fused in order to compute a single value. ACE uses the weighted mean

as a fusion scoring method:

$$s = \sum_{i=1}^{m} w_i s_i \quad \text{with} \quad \sum w_i = 1 \tag{5.2}$$

where m is the number of computed scores for that element.

The value v of a CSS property is then modified based on the following style function:

$$v = v(1 + s) \tag{5.3}$$

On subsequent access to the UI, the new scores s'_i and how they will affect the CSS properties are both updated as follows:

$$\begin{aligned} s'_i &= \zeta(n'_i/N) - s_i \\ v' &= v(1 + s') \end{aligned} \tag{5.4}$$

According to equations (5.3) and (5.4), when a user loads an application for the first time, elements are rendered as they were designed, as the system has no information about previous interactions ($s_i = 0 \ \ \forall i$). Then, when returning to the application the system will react accordingly, i.e., modifying the value of those CSS properties specified by the webmaster based on the amount of user's interactions.

Given that scores are bounded to the interval $(-1, 1)$, a score of, say, 0.05 for a `margin-top` property will be interpreted as "increasing by 5% the value of the top margin." Conversely, a score of -0.1 for a `color` property will be interpreted as "decreasing by 10% (the contrast or saturation of) the font color." This way, it is not possible to alter the visual properties significantly, since adaptations are incrementally applied. Event lists are the only user information stored in the local database.

5.4 Fostering Creativity

ACE also introduces an interesting framework to find inspirational examples for redesigning UIs. Typically, the primary purpose of prototyping tools is to provide feedback to define a design earlier, when there is inadequate information to choose one solution over another. However, once the design of an application or website leaves the testing phase and moves to production, it hardly ever gets substantially modified. Rather, it follows a cycle of subtle iterative improvements. At this stage, surprisingly, few methods seldom support incrementally revisiting different versions of the *same* solution.

In this line, some work has been done in generating design *alternatives* to assist the user in the design process, i.e., to get the "right design", for instance,

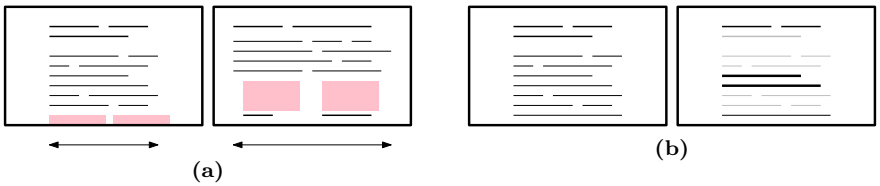


Figure 5.7: Some redesign considerations. [5.7a] Widening the central column of a web page allows the browser to display more information at a glance. [5.7b] Some parts of the UI can be altered according to its importance; e.g., changing the font sizes and colors of headings and text paragraphs.

Design Gallery [Marks et al., 1997], Side Views [Terry and Mynatt, 2002], or Adaptive Ideas [Lee et al., 2010]. However, there is little research toward tools that allow designers to explore design *refinements*, i.e., to get the "design right". Traditionally, current techniques to suggest improvements to an existing design imply a manual implementation (see Section 5.2). What would be interesting, though, is being able to automate the process to a greater or a lesser extent. In this regard, Masson et al. [2010] proposed using interactive genetic algorithms to add permutations to an existing design. The downside of this approach is that it relies on a user-task model and therefore it must be learned. In contrast, we propose to use ACE, which is model-free, and lets *all* users take part in the design process. However, instead of adapting a UI to an individual, the interactions of all users can be exploited to alter the design of an application or a whole website. Among other benefits, this may allow designers to:

1. Avoid having to recruit users for testing each time the application is updated: what you see is what users do.
2. Discover visually what behavioral patterns are consensus.
3. Find inspirational examples, by looking at how the appearance of the UI gets modified over time.

If subtle design modifications are needed to refine an existing layout—as it often happens when iterating over a design solution—then implicit user interaction can be valuable to this end [Leiva, 2012b]. For instance, on websites, if all users spend most of their browsing time on the home page 'above the fold', the designer could consider make wider the main body content, so that some parts could be accessed faster (Figure 5.7a). Similarly, if there is some paragraph that is commonly selected, if would be interesting to make such text more prominent, probably by increasing the font size or the color contrast, so that in subsequent visits users could realize easily where is the popular information (Figure 5.7b).

We believe therefore that ACE can exploit the collective users' behavior as an inspirational source for UI redesign. Implicit interactions can be gathered at scale on a daily basis, and without burdening the user. What is more, on the

Web, independent feedback is received from hundreds or thousands of remote anonymous users rather than being produced and interpreted in a small group or individuals working in isolation. This may help to achieve (hopefully) better design decisions, since it is possible to empirically validate how users react to a particular design update; e.g., by carrying out A/B tests. Additionally, this has the notable advantage that data acquisition and later processing can be both completely automated.

5.5 Evaluation

In terms of system performance, ACE takes a few milliseconds to complete the adaptation process. A series of JavaScript benchmarks were performed on the sample page shown in Figure 5.3 with different configuration objects and CSS properties. The machine was an i686 @ 2 GHz with 1 GB of RAM. The adaptation code was executed 100 times and benchmark results were averaged. Concretely, for 10 items (that were specified by different CSS level 3 selectors[4]) having at most 5 properties each, in all tested browsers (Firefox 7, Chrome 15, Opera 11, Internet Explorer 9, and Dolphin 2.2) the average times were below 20 ms, with standard deviations below 0.1 in all cases.

Regarding human evaluation, devising the most suitable evaluation method is still not completely clear. As a preliminary approximation, an informal study involving 12 users was carried out, in which participants where told to freely browse a mockup site (Figure 5.1) with the ACE system on an HTC Desire [Leiva, 2011]. At the end of the test, users answered three questions (see Figure 5.8); **Q1**: Do you think page elements are well laid out? **Q2**: Did you notice any change on the page, regarding the first time you visited it? **Q3**: If so, did you find distracting those changes?

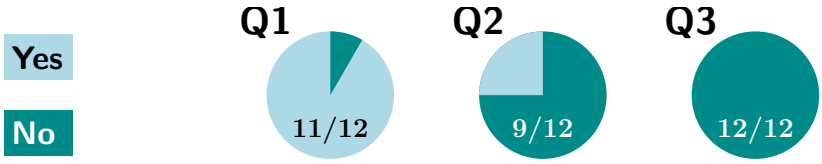


Figure 5.8: Results of the informal user questionnaire.

Overall, users' acceptability toward the method was perceived as positive. As observed, nine of them did not notice the automatic modifications, and none found distracting those changes while browsing. The informal pilot study, although being not conclusive, revealed that this adaptation technique has an interesting potential in building adaptive user interfaces.

[4] http://www.w3.org/TR/css3-selectors/

Regarding using ACE as a source of creative redesign, previous informal meetings with web designers have shown that this tool is perceived as a useful help [Leiva, 2012b]. People commented that they often want to determine how changes to a few page elements will affect the final appearance of the website. ACE satisfies this need, by letting them to inspect how user behavior would influence CSS rendering. Moreover, automatic redesign frees the web designer from the need to know what changes are possible, or how they can be effectively performed. Also, design refinements can offer pragmatic value as well as inspirational value. Figure 5.9 depicts some examples that this tool can produce.

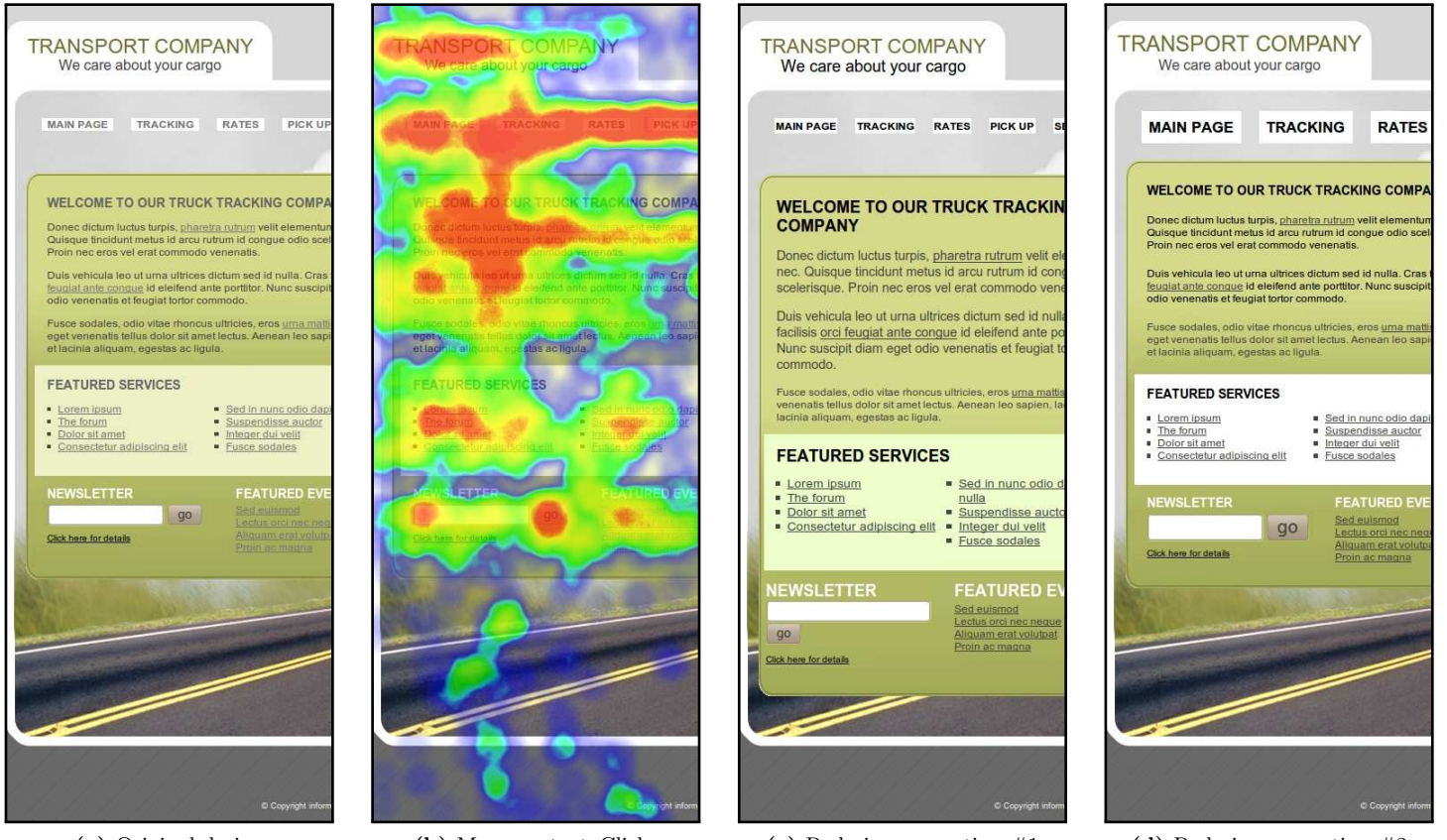
5.6 Discussion

Automatically mining implicit interactions for UI adaptation *and* redesign is a promising direction for future research. However, some work still remains to be done.

First of all, we feel that evaluating this kind of adaptation strategy is quite challenging, since no objective metrics can be consistently computed; e.g., in the absence of labeled samples, we cannot apply well-known measures such as precision and recall; and having to interrupt the normal navigation flow of users to ask them to vote is certainly not an option. We strongly believe, though, that implicit interactions inherently encode performance. Thus, if an adapted design works better than a previous iteration, it should be reflected somehow in the traces of movements, gestures, etc. Nevertheless, one needs to be cautious with this hypothesis, since learnability and familiarity with the UI could be introducing a serious bias. Therefore, an immediate follow-up work will consist in carrying out a formal in-lab evaluation study.

On the other hand, since content is automatically generated, it is likely to be of less quality than human-generated content. Thus, we believe that it would be interesting to assess the influence of such variations in layout design, or use different evaluation viewpoints; e.g., measure the reduction of user effort, compare to other adaptive systems, etc.

ACE has some implications for participatory design as well, since it aims to create applications that are more appropriate to their users. As previously commented, this frees the UI designer from the need to know what changes are possible; but more importantly, it helps to determine *how* such changes can be effectively performed. Also, system suggestions are expected to offer pragmatic value as well as inspirational value to the designer. ACE also can contribute to find "interaction agreements" between all users, which may be useful to detect whether if a design works as expected; e.g., how designs change through time according to the heterogeneous behavior of the users. Additionally, non-experienced designers can gain insights about what is going on with



(a) Original design **(b)** Movements + Clicks **(c)** Redesign suggestion #1 **(d)** Redesign suggestion #2

Figure 5.9: Redesign examples produced by ACE, taking into account multiple interaction logs and overriding a few CSS rules.

their designs, from the user interactions' point of view. This suggests implications for design practices from which the HCI community may well be able to benefit. Finally, collected data can be reused to support design decision making, or to improve understanding of how users interact at scale. Data can also be used for complementary analytics in traditional usability tests, or applied to infer new knowledge for future users.

A known limitation of ACE is that currently it can adapt only those properties that vary in a numerical range; e.g., `max-height` or `padding`. However, in a future it is expected to be able to map semantic properties. For instance, to adapt the `text-align` property of a text paragraph one could use:

$$v = \begin{cases} \texttt{"left"} & \text{if } s \in (-1, -0.5] \\ \texttt{"center"} & \text{if } s \in (-0.5, 0.5) \\ \texttt{"right"} & \text{if } s \in [0.5, 1) \end{cases}$$

Finally, redesign decisions are (by now) based on modifications of shape, position, and/or color attributes. Therefore, more advanced adaptation strategies such as re-arranging several page elements (beyond alignment) or inserting/removing content would require a technically more sophisticated approach.

All in all, this technology enables a straightforward means to invisibly enhance the utility of regular applications and web pages; e.g., in terms of usability, accessibility, readability, interactivity, or performance. Systems like ACE may allow applications to be flexible enough to meet different user needs, preferences, and situations.

5.7 Conclusions and Future Work

Dynamic and continuously changing environments like the Web demand new means of building UIs that are aligned to the skills of the users. We have presented an alternative to redesign interface widgets that operates unobtrusively for both 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.

Finally, we believe that this work opens a door to a wealth of applications that can be developed by tracking the user activity and dynamically restyling the appearance of the UI in response. For instance, integrating ACE with an eye-tracker would provide a finer-grained and potentially more focused analysis of user interactions. Moreover, other biometric inputs such as electrocardiogram signals would allow developers create "organic" UIs that are able to react to the emotions of the users.

Further research will pursue more ambitious results, such as inferring high-level behaviors from low-level events—for instance, reporting if a certain design causes users to get lost or incites them to being more active.

Bibliography of Chapter 5

E. Arroyo, T. Selker, and W. Wei. Usability tool for analysis of web designs using mouse tracks. In *Proceedings of extended abstracts on Human factors in computing systems (CHI EA)*, pp. 484–489, 2006.

R. Atterer, M. Wnuk, and A. Schmidt. Knowing the user's every move – user activity tracking for website usability evaluation and implicit interaction. In *Proceedings of the 15th international conference on World Wide Web (WWW)*, pp. 203–212, 2006.

N. Bila, T. Ronda, I. Mohomed, K. N. Truong, and E. de Lara. PageTailor: reusable end-user customization for the mobile web. In *Proc. MobySys*, pp. 16–29, 2007.

M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller. Automation and customization of rendered web pages. In *Proceedings of the 18th annual ACM symposium on User interface software and technology (UIST)*, pp. 163–172, 2005.

M. Claypool, P. Le, M. Wased, and D. Brown. Implicit interest indicators. In *Proceedings of the 6th international conference on Intelligent user interfaces (IUI)*, pp. 33–40, 2001.

L. Findlater and J. McGrenere. Impact of screen size on performance, awareness, and user satisfaction with adaptive graphical user interfaces. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems (CHI)*, pp. 1247–1256, 2008.

K. Z. Gajos and D. S. Weld. SUPPLE: Automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces (IUI)*, pp. 93–100, 2004.

K. Z. Gajos, J. O. Wobbrock, and D. S. Weld. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of the 20th annual ACM symposium on User interface software and technology (UIST)*, pp. 231–240, 2007.

K. Z. Gajos, K. Everitt, D. S. Tan, M. Czerwinski, and D. S. Weld. Predictability and accuracy in adaptive user interfaces. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems (CHI)*, pp. 1271–1274, 2008.

M. Y. Ivory and M. A. Hearst. Statistical profiles of highly-rated web sites. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI)*, pp. 367–374, 2002.

S. Kurniawan, A. King, D. Evans, and P. Blenkhorn. Personalising web page presentation for older people. *Interacting with Computers*, 18(3):457–477, 2006.

B. Lee, S. Srivastava, R. Kumar, R. Brafman, and S. R. Klemmer. Designing with interactive example galleries. In *Proceedings of the 28th international conference on Human factors in computing systems (CHI)*, pp. 2257–2266, 2010.

L. A. Leiva. Restyling website design via touch-based interactions. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI)*, pp. 91–94, 2011.

L. A. Leiva. Interaction-based user interface redesign. In *Proceedings of the 17th international conference on Intelligent User Interfaces (IUI)*, pp. 311–312, 2012a.

L. A. Leiva. Automatic web design refinements based on collective user behavior. In *Proceedings of the 2012 annual conference extended abstracts on Human factors in computing systems (CHI EA)*, pp. 1607–1612, 2012b.

J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, pp. 389–400, 1997.

D. Masson, A. Demeure, and G. Calvary. Magellan, an evolutionary system to foster user interface design creativity. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems (EICS)*, pp. 87–92, 2010.

M. Terry and E. D. Mynatt. Side views: Persistent, on-demand previews for open-ended tasks. In *Proceedings of the 15th annual ACM symposium on User interface software and technology (UIST)*, pp. 71–80, 2002.

T. Tsandilas and M. C. Schraefel. User-controlled link adaptation. In *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia (HT)*, pp. 152–160, 2003.

D. Zhang. Web content adaptation for mobile handheld devices. *Communications of the ACM*, 50(2):75–79, 2007.