Cross-Device Consistency in Automatically Generated User Interfaces

Krzysztof Gajos, Anthony Wu and Daniel S. Weld
University of Washington
Seattle, WA, USA
{kgajos,anthonyw,weld}@cs.washington.edu

INTRODUCTION

The growing importance of ubiquitous computing has motivated an outburst of research on automatic generation of user interfaces for different devices (*e.g.*, [6] or our own SUPPLE [4]). In some cases, care is taken to ensure that similar functionality is rendered similarly across different applications on the same device [5]. However, we also need to ensure that after using an application on one device (say, a PDA) and having learned that user interface, the user will not have to expend much effort having to learn a brand-new user interface for the same application when moving to a new platform (*e.g.*, a touch panel). We have began to extend our SUPPLE system in a way that allows it to produce interfaces that make a trade off between optimality given a new platform and similarity to the previously rendered user interfaces for the same application. In particular:

- we show how to incorporate an interface dissimilarity metric into a UI generation process resulting in new interfaces resembling ones previously used by the user;
- we propose a list of most salient widget features that can be used to asses similarity of interfaces rendered on radically different platforms;
- and we outline the most promising approaches for automatically learning parameters of a UI dissimilarity function from user feedback.

INTERFACE GENERATION AS OPTIMIZATION

We cast the user interface generation and adaptation as a decision-theoretic optimization problem, where the goal is to minimize the estimated user effort for manipulating a candidate rendering of the interface. SUPPLE takes three inputs: a *functional interface specification*, a *device model* and a *user model*. The functional description defines the *types* of data that need to be exchanged between the user and the application. The device model describes the widgets available on the device, as well as cost functions, which estimate the user effort required for manipulating supported widgets with the interaction methods supported by the device. Finally, we model a user's typical activities with a device- and rendering-independent *user trace*. Details of these models and rendering algorithms are available in [4].

We have now extended our cost function to include a measure of dissimilarity between the current rendering ϕ and a previous *reference* rendering ϕ_{ref}:

$$\$(\phi, \mathcal{T}, \phi_{ref}) = \$(\phi, \mathcal{T}) + \alpha_s \mathcal{S}(\phi, \phi_{ref})$$

Here, $\mathcal{T}$ stands for a user trace (which allows SUPPLE to personalize the rendering), $\$(\phi, \mathcal{T})$ is the original cost function (as in [4]) and $\mathcal{S}(\phi, \phi_{ref})$ is a dissimilarity metric. The user-tunable parameter α_s controls the trade-off between a design that would be optimal for the current platform and one that would be maximally similar to the previously seen interface (see Figure 1).

We define the dissimilarity metric as a linear combination of K *factors* $f^k : \mathcal{W} \times \mathcal{W} \mapsto \{0, 1\}$, which for any pair of widgets return 0 or 1 depending on whether or not the two widgets are similar according to a certain criterion. Each factor corresponds to a different criterion. To calculate the dissimilarity, we iterate over all elements e of the functional specification $\mathcal{E}$ of an interface and sum over all factors:

$$\mathcal{S}(\phi, \phi_{ref}) = \sum_{e \in \mathcal{E}} \sum_{k=1}^{K} u_k f^k(\phi(e), \phi_{ref}(e))$$

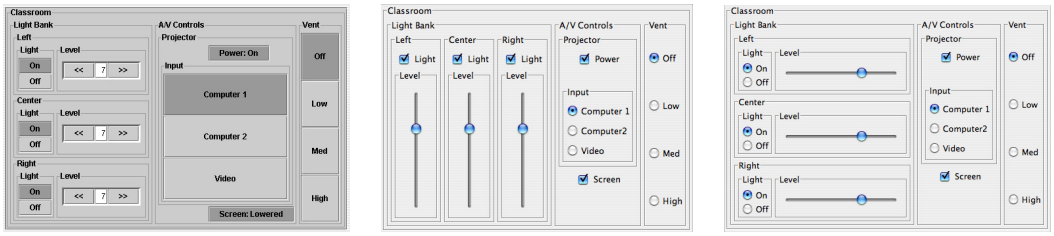
In the following two sections we will discuss what widget features we have identified as good candidates for constructing the factors and how we can learn their relative weights u_k.

RELEVANT WIDGET FEATURES

To find the relevant widget features for comparing interface renderings across different platforms, we generated interfaces for several different applications for several different platforms and we picked sets that we considered most similar. We have identified a number of widget features sufficient to explain all the results we generated. The following are the features of primitive widgets (*i.e.*, widgets used to directly manipulate functionality):

Language {toggle, text, position, icon, color} – the primary method(s) the widget uses to convey its value; for example, the slider uses the position, list uses text and position, checkbox uses toggle.

Domain visibility {full, partial, current value} – some widgets, like sliders, show the entire domain of possible values, lists and combo boxes are likely to show only a subset of all possible values while spinners only show the current value.



(a) (b) (c)

Figure 1: A basic example: (a) a reference touch panel rendering of a classroom controller interface, (b) the rendering SUPPLE considered optimal on a keyboard and pointer device in the absence of similarity information, (c) the rendering SUPPLE produced with the touch panel rendering as a reference (the dissimilarity function parameters were set manually).

Orientation of data presentation {vertical, horizontal, circular} – if the domain of possible values is at least partially visible, there are different ways of arranging these values.

Continuous/discrete – indicates whether or not a widget is capable of changing its value along a continuous range (*e.g.*, a slider can while a list or a text field are considered discrete).

Variable domain {yes, no} – the domain of possible values can be easily changed at run time for some widgets (*e.g.*, lists), while it is not customary to do it for others (*e.g.*, sets of radio buttons).

Primary manipulation method {point, type, drag} – the primary way of interacting with the widget.

Widget geometry {vertical, horizontal, even} – corresponds to the general appearance of the widget.

We will omit here the features of container widgets (*i.e.*, those used to organize other elements) because they mostly have to do with obvious widget properties, such as the layout and visibility of sub elements.

LEARNING THE DISSIMILARITY METRIC

We aim to find values of the parameters u_k for the dissimilarity metric that best reflect the user's perception of user interface similarity. We propose to do it by automatically learning these parameters by asking user explicit binary queries (*i.e.*, "which of the two interfaces looks more like the reference rendering?"). We will learn rough estimates of these parameters by eliciting responses from a significant number of users in a controlled study. This will allow SUPPLE to behave reasonably "out of the box" while still making it possible for individual users to further refine the parameters. We are thus looking for a computationally efficient learning method that will allow SUPPLE to learn from a small number of examples and that will support efficient computation of optimal or near optimal binary queries.

One very elegant approach to this problem is to treat the parameters u_k as random variables [2], whose estimates are updated in response to the gathered evidence by inference in a Bayes network. This approach makes it very easy to encode prior knowledge and it provides an intuitive mechanism for integrating accumulating evidence. However, there is no compact way to represent the posterior distribution using this approach so, in theory, it may be necessary to keep a full log of all of user's feedback and re-sample the model after each new piece of evidence is obtained. Also, it is notoriously hard to compute optimal queries to ask of the user when reasoning about the expected value of the target function (although efficient methods have been found for some well defined domains, *e.g.* [3]).

Methods based on *minimax regret* allow the factors u_k to be specified as intervals and learning proceeds by halving these intervals on either side in response to accumulated evidence. These methods are particularly attractive because computationally efficient utility elicitation methods have been developed within this framework [1]. The main drawback of this approach is that it is not robust in the face of inconsistent feedback from the user.

An algorithm based on a standard method for training support vector machines has been proposed for learning distance metrics from relative comparisons [7]. This method may likely produce the best results, although an efficient method would need to be developed for generating optimal queries so that appropriate training data could be obtained with minimal disturbance to the user.

Acknowledgments This research is supported by NSF grant IIS-0307906, ONR grant N00014-02-1-0932 and by DARPA project CALO through SRI grant number 03-000225. Thanks to Batya Friedman and her group for useful discussion, and to Anna Cavender for comments on the manuscript.

REFERENCES

1. C. Boutilier, R. Patrascu, P. Poupart, and D. Schuurmans. Regret-based utility elicitation in constraint-based decision problems. Working Paper.
2. U. Chajewska and D. Koller. Utilities as random variables: Density estimation and structure discovery. In *UAI'00*, 2000.
3. U. Chajewska, D. Koller, and R. Parr. Making rational decisions using adaptive utility elicitation. In *AAAI'00*, 2000.
4. K. Gajos and D. S. Weld. Supple: automatically generating user interfaces. In *IUI'04*, Funchal, Portugal, 2004.
5. J. Nichols, B. A. Myers, and K. Litwack. Improving automatic interface generation with smart templates. In *IUI'04*, 2004.
6. S. Nylander, M. Bylund, and A. Waern. The ubiquitous interactor - device independent access to mobile services. In *CADUI'04*, Funchal, Portugal, 2004.
7. M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *NIPS'03*, 2003.