Automatically Generating User Interfaces For Ubiquitous Applications

Krzysztof Gajos and Daniel S. Weld

University of Washington
Seattle, WA, USA
{kgajos,weld}@cs.washington.edu

Abstract. SUPPLE is an application and device-independent system, currently under development at University of Washington, that automatically generates user interfaces for a wide variety of display devices. SUPPLE uses decision-theoretic optimization to render an interface from an abstract *functional specification* and an interchangeable *device model*. SUPPLE also provides adaptation and customization mechanisms that allow for system- and user-initiated modifications to the appearance, organization and navigational structure of the user interface.

1 Introduction

Ubiquitous computing promises seamless access to a wide range of computational tools and Internet-based services — regardless of the user's physical location. For example, when a person enters a room, she may want to control the room's equipment and applications from her PDA. Alternatively, the user may be carrying a display-less device, such as the Personal Server [8], and want to use any display available (such as her phone, a friend's laptop or a public touch panel) to interact with the data and applications stored on her Personal Server.

A critical aspect of this vision is the premise that every application (whether it be an email client or room-lighting controller) should be able to render an interface on any device at the user's disposal. Given the wide range of device types, form factors, and input methods, it is unscalable for the human programmers to create interfaces for each type of device. Instead, an automated solution is necessary. This is even more apparent when we consider the fact that sometimes new functionality emerges from runtime combinations of available applications and appliances [5].

Essential to our long-term vision [9] is also the observation that today's software is mass produced with a plethora of features designed to address the needs of very diverse user populations. The interfaces shipped with popular applications are designed in a "one size fits all" manner, but by aiming to address the needs of the "average user" they miss *essential* needs of almost every individual user. In contrast, wouldn't it be ideal if every user could have a custom-built UI that best reflected his or her needs? In this paper, however, we will focus primarily on the challenge of adapting the user interfaces to the wide variety of display devices available in today's ubiquitous computing environments.

A number of researchers have identified this challenge and several solutions have been proposed, *e.g.* [3, 5, 6]. Although promising, the current solutions do not handle device constraints in a general enough manner to accommodate the wide range of display sizes and interaction styles available even in today's ubicomp environments.

With the SUPPLE system, we take a different approach — treating interface generation as an optimization problem. When asked to render an interface (specified functionally) on a specific device and for a specific user, SUPPLE searches for the rendition that meets the device's constraints and minimizes the estimated cost (user effort) of the person's activity. For example, Figure 2 in Section 4 depicts two different interfaces SUPPLE deemed optimal (with respect to a cost function derived from the device and user models) for two devices with the same screen size but different interaction methods. It is important to note that unlike some of the earlier work, *e.g.* [4, 1], we not only compute the optimal layout but also choose the individual widgets to be used in rendering of the UI and the overall navigation structure of the final interface (*i.e.*, placement of parts of the interface in tab panes, pop up windows, etc.).

2 User Interface Generation As Optimization

We cast the user interface generation 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*.

The information provided by these models is combined by our constraint-based branch-and-bound search algorithm. At each step of the search, the algorithm attempts to assign a widget to an element in the interface specification. Primitive elements (those corresponding to actual functionality in the application) are paired with widgets that are capable of displaying (or manipulating) information of the element's type. Container elements (those that provide logical groupings within the interface model) are paired with different layout panes (vertical, horizontal, grids), tab panes, etc. There are usually a number of different widgets that could be assigned to any element in the interface specification. The objective is to find such a global assignment, that results in the "best" concrete interface rendering that satisfies all device constraints (such as the screen size). The "goodness" of the resulting interface rendering is computed based on the cost functions included in the device specification and the traces contained in the user model.

The main algorithmic challenge stems from the fact that the number of possible combinations of widget assignments to interface elements grows exponentially with the number of elements in the interface description. For example, there were 1.8×10^9 possible assignment combinations for the classroom controller when rendered on a keyboard and pointer device (Figure 2) but our algorithm was able to prune a vast majority of the search space and, consequently, the time necessary to produce the interfaces ranges from a fraction of a second to 2 seconds on a standard desktop computer. The computational requirements of our algorithm make it impractical to run it on an impoverished platform such as a PDA or a cellular phone. We discuss the practical implications of this fact in the next section. Details of the models and rendering algorithms are available in [2].

3 Architecture

The three models that Supple combines to create a concrete user interface, come from three independent sources:

- the user model is stored in user's personal information space;
- the interface model is provided by the application or appliance developers;
- the device model, describing the capabilities of the display device, is provided by the manufacturer of that device.

In ubiquitous computing, where users, applications and devices are all nomadic, and where spontaneous interactions are a norm, we need to be prepared for the "worst case" scenario, where each of the three models resides on a different physical device and none of the devices is powerful enough to host the computation necessary to render the interface. To make it more concrete, consider a simple scenario, where a user enters a public space, which is equipped with a music system, and the user wishes to play some music and control it from his PDA. His user model is stored on his Personal Server [8] – a small portable computer without its own display; the device model comes from the PDA, and the UI model is provided by the music system. None of these devices is capable of performing the computation required to render the interface quickly enough, so another, faster, computer has to be employed to run the Supple algorithm.

These considerations dictate how our system has to be modularized. Although other considerations have prevented us from investing the necessary effort to fully distribute our system, we have enforced strict abstraction barriers so as to allow for distribution at a later time. Figure 1 illustrates the components of the Supple system and the flow of information between them. Solid arrows show information flow during the rendering process while the dashed lines show the data transfered while the user interacts with the rendered interface. First, the three models need to be sent to Supple (arrows 1-3) and the rendered interface description is sent to the target device (arrow 4). While the user interacts with the interface, his actions are sent to the application or appliance, while the changes in the state of the latter are communicated back to the interface (arrow 5). Finally, all the interactions are sent to the user's personal device for recording in the user model (arrow 6).

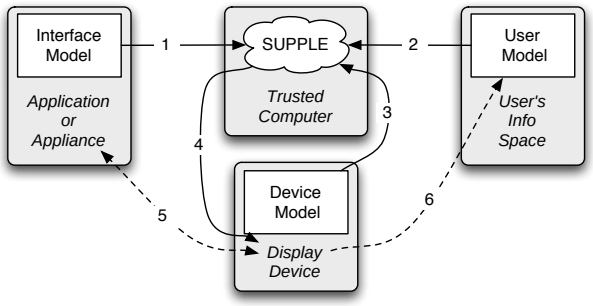

Fig. 1. SUPPLE architecture. 1-3. Models (for the Interface, User and Device) are sent to SUPPLE; 4. Generated interface is set to the Display Device; 5. The Display Device communicates user's actions to the Application or Appliance, while the latter sends its state updates back to the device to be reflected in the UI; 6. The Display Device sends the log of user's actions to the user's personal device for inclusion in the User Model. In some situations, it may be more convenient to route communications along arrows 5 and 6 through SUPPLE rather than establish additional direct connections.

4 Implementation Status

In [2] we demonstrated a working version of SUPPLE, that generates UIs for three different platforms: keyboard and pointer devices, touch panels and WAP cell phones. More recently, we generalized our architecture to render interfaces with cyclic functional specifications; this allows SUPPLE to render personalized interfaces with shortcuts to frequently accessed functionality. We have also created support for user-initiated customizations (e.g, deleting, moving or duplicating arbitrary functionality throughout the rendered interface). Finally, we demonstrated SUPPLE's scalability by implementing two larger applications: an email client and a distributed jukebox, which allows a group of individuals to collaboratively manage a shared playlist on a common stereo.

Figures 2 and 3 show an interface for a classroom controller rendered on three different types of devices: a keyboard and pointer device, a touch panel and a WAP cell phone. Figure 4 shows an interface to a three-component stereo system rendered on two keyboard and pointer devices of different sizes. As was mentioned above, more complex applications have been rendered using SUPPLE but space constraints preclude inclusion of their interfaces in this paper.

5 Related Work

Our work builds on earlier research on model-based user interface generation [7], but differs in several important aspects. Most importantly, in contrast to the rule-based approaches used previously, including systems such as the Personal

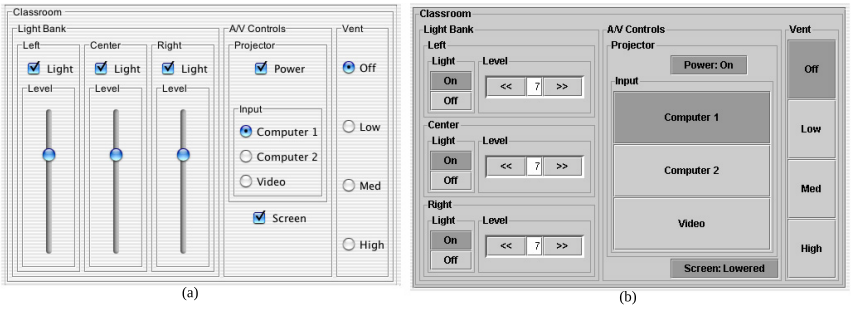

Fig. 2. The classroom interface rendered for two devices with the same size: (a) a pointer-based device (b) a touch-panel device

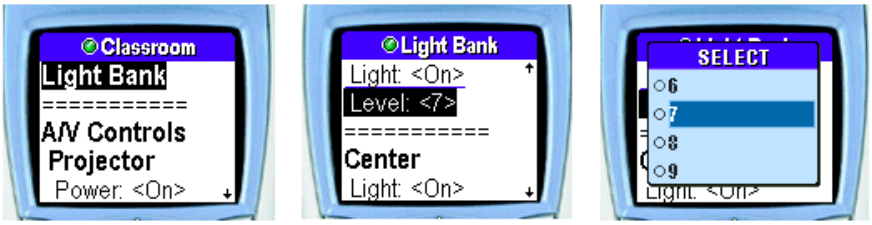

Fig. 3. The classroom interface rendered on a WAP cell phone simulator (Sony Ericsson T68i); the successive screen shots illustrate the steps necessary to manipulate the brightness level of one of the lights.

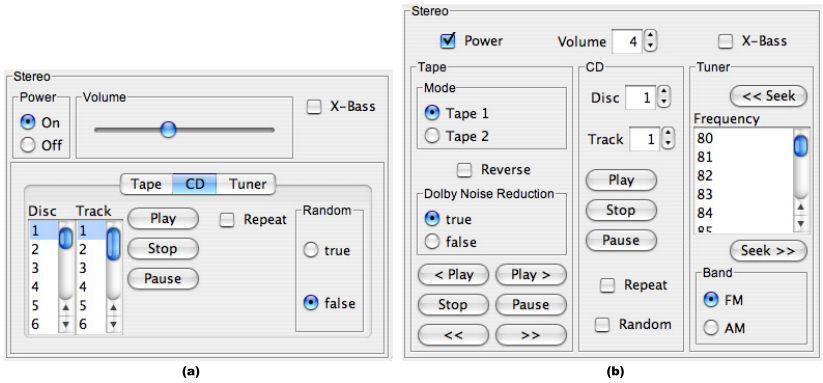

Fig. 4. An interface to a stereo system rendered on two keyboard and pointer devices of different sizes.

Universal Controller (PUC) [3], we use optimization to select widgets, design the navigation structure (decide if any parts of the interface should be placed in tab panes or popup windows) and lay out the elements. Other modern user interface generation system differ from ours in that, for example, iCrafter [5] relies on hand-crafted templates and XIML [6] relies on the interface designer to explicitly specify what widgets to use under what size constraints.

6 Conclusions

The success of our system ultimately depends on its adoption. On the one hand, the interfaces we generate need to adequately address users' needs. On the other hand, we are aware that designers will be initially very uncomfortable using our functional specification and relinquishing control over the final appearance of the interfaces. To address the concerns of the users, we are currently working on powerful adaptation and customization support that will provide users with highly individualized interfaces. The designers' concerns may lead to a hybrid approach, where manually created renderings will be provided for a small number of most popular platforms, while automatically generated interfaces will be provided in other situations.

Acknowledgments This work supported by NSF grant IIS-0307906 and ONR grant N00014-02-1-0932. Raphael Hoffman is working on the customization capabilities of SUPPLE. Thanks to Gaetano Borriello and James Landay.

References

1. J. Fogarty and S. E. Hudson. GADGET: A toolkit for optimization-based approaches to interface and display generation. In *UIST'03*, Vancouver, Canada, 2003.
2. Krzysztof Gajos and Daniel S. Weld. Supple: automatically generating user interfaces. In *IUI'04*, Funchal, Portugal, 2004.
3. Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joe Hughes, Thomas K. Harris, Roni Rosenfeld, and Mathilde Pignol. Generating remote control interfaces for complex appliances. In *UIST'02*, Paris, France, 2002.
4. Dan R. Olsen, Sean Jefferies, Travis Nielsen, William Moyes, and Paul Fredrickson. Cross-modal interaction using XWeb. In *UIST'00*, pages 191–200, San Diego, California, United States, 2000.
5. S. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. ICrafter: A service framework for ubiquitous computing environments. In *Ubicomp 2001*, 2001.
6. Angel Puerta and Jacom Eisenstein. XIML: A universal language for user interfaces, 2002. unpublished paper available at http://www.ximl.org/.
7. Pedro Szekely. Retrospective and challenges for model-based interface development. In F. Bodart and J. Vanderdonckt, editors, *Design, Specification and Verification of Interactive Systems '96*, pages 1–27, Wien, 1996. Springer-Verlag.
8. Roy Want, Trevor Pering, Gunner Danneels, Muthu Kumar, Murali Sundar, and John Light. The personal server: Changing the way we think about ubiquitous computing. In *Ubicomp 2002*, Goteborg, Sweden, 2002.
9. Daniel S. Weld, Corin Anderson, Pedro Domingos, Oren Etzioni, Krzysztof Gajos, Tessa Lau, and Steve Wolfman. Automatically personalizing user interfaces. In *IJCAI03*, Acapulco, Mexico, August 2003.