Automatically Generating User Interfaces for Appliances

Jeffrey Nichols
Human Computer Interaction Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213 USA
jeffreyn@cs.cmu.edu

ABSTRACT
Today's complex appliances are plagued by difficult-to-use and inconsistent user interfaces. I am building the *personal universal controller* (PUC) system that addresses these problems by separating the user interface from the appliance. Users will control their appliances using personal devices they already have, such as personal digital assistants (PDAs) or mobile phones. User interfaces are automatically generated so that each appliance interface can be customized for the user and the device on which the interface is displayed. Interfaces generated by the PUC system will take into account interfaces previously generated for the user and create single combined interfaces for multiple connected appliances, like a home theater, that usually have separate interfaces for each appliance. I intend to evaluate the completed system by conducting user studies to show that my automatically generated interfaces are more usable than functionally identical manufacturers' interfaces.

Categories and Subject Descriptors: D.2.2 [**Design Tools and Techniques**]: User interfaces—*automatic generation*. H.5.2 [**User Interfaces**]: Graphical user interfaces (GUIs), Voice I/O—*handheld computer interfaces, speech user interfaces.*

Additional Keywords and Phrases: Automatic interface generation, Pebbles, appliances, personal digital assistants (PDAs), personal universal controller (PUC)

INTRODUCTION

Users interact daily with many computerized appliances at their homes and offices, including media players, kitchen appliances, copiers, etc. The number and complexity of these appliances is increasing as the cost of microprocessors decreases. Unfortunately, the trend has been that as appliances get more computerized with more features, their user interfaces become harder to use [1].

I am exploring a solution to this problem that moves the user interface from the appliance to a separate device that specializes in providing user interfaces, i.e. a "UI device." Personal digital assistants (PDAs) and mobile phones are examples of devices that might be used as UI devices. Many phones and PDAs available today have the ability to communicate with appliances through wireless networking protocols like 802.11 (Wi-Fi) or Bluetooth. Furthermore, most phones and PDAs are built with specialized interface hardware, like color and/or touch-sensitive LCD screens, which make the creation of high quality user interfaces easier. A phone or PDA could leverage its specialized hardware to provide better user interfaces than what can be built cost-effectively into an appliance.

My approach is called the Personal Universal Controller (PUC) [5, 7][1]. PUC UI devices automatically generate user interfaces for the complete functionality of appliances such as stereos, copiers, elevators, and the non-driving functions of a car. Interfaces can be generated for multiple platforms and modalities: graphical interfaces can be created on PDAs, mobile phones, and desktop computers, and speech interfaces can be created with the Universal Speech Interfaces framework [12].

RELATED WORK

Automatic user interface generation has been investigated by many researchers in the past [13]. Unlike the PUC system, most other work relied on an interaction designer to guide generation and/or to edit the resulting interfaces to fix any design problems. End-users of the PUC system will not be willing to spend the time and effort to modify their user interfaces in these ways, and thus the PUC system must generate high quality user interfaces on the first attempt. Previous work in automatically generating dialog boxes suggests that this goal is plausible [4], and I am developing new techniques that increase the likelihood that high quality interfaces will be generated without designer intervention.

There has also been significant work in appliance control by both industrial and academic groups. UPnP [14] and HAVi [2] are both consortiums of consumer electronics companies that are building technologies to unify the control of electronic appliances. This work will make more appliances controllable and will help the vision of a PUC succeed. However, these projects focus on the infrastructure and have not examined many interface issues.

Researchers have also examined how appliances can be controlled. Systems such as the Universal Interactor [3] and

UIST '04, October 24–27, 2004, Santa Fe, New Mexico, USA
ACM 1-58113-962-4/04/0010

[1] This paper is adapted from material previously published in [5].

ICrafter [11] are investigating infrastructures to move the appliance user interfaces to controller devices. Though both of these systems support simple automatic generation of user interfaces, both prefer to use hand-generated interfaces when available. The PUC system differs from these systems in its focus on automatic user interface generation and the quality of the resulting interfaces. Xweb [9] is another research project that automatically generates interfaces for interactive services from an abstract description. The PUC system extends the ideas of Xweb with support for ensuring consistent user interfaces across similar appliances and the ability to generate a single interface for controlling multiple appliances.

HAND-DESIGNED INTERFACES AND USER STUDIES

Although automatically generating high-quality interfaces seems plausible, it is also a hard problem. No previous system has successfully generated user interfaces measured to be of high-quality. The problem can be broken down into two sub-problems: determining what information an abstract appliance specification should include, and building an interface generator that can design high-quality interfaces from those abstract specifications. As a starting point, I hand-designed remote control interfaces for several appliances (rather than begin with designing a language for describing the appliances). Then I conducted user studies to compare the hand-designed interfaces to the manufacturers' interfaces, where I found that subjects using my hand-designed interfaces were twice as fast and made half as many errors as subjects using the manufacturers' interfaces [6]. This approach allowed me to concentrate on what functional information about the appliance is necessary to create a usable interface and to show that a PUC controller could be easier to use than interfaces on actual appliances.

PUC ARCHITECTURE

The PUC is designed to allow users to control appliances in their environment through a remote user interface. The remote control might be a graphical user interface on a PDA or mobile phone, or it could be a speech interface that uses microphones in the room. When a user decides to control an appliance, the controller device would download from that appliance an abstract functional description and use that description to automatically generate an interface for controlling that appliance[2]. A two-way communication channel between the controller and the appliance allows the user's commands to be sent to the appliance and feedback to be provided to the user.

The PUC system has four parts: a specification language, a communication protocol, appliance adaptors, and interface generators. All of these pieces are described in more detail elsewhere [7]. Automatic generation of user interfaces is enabled by the specification language, which allows each appliance to describe its functions in an abstract way. The goal in designing this language was to include enough information to generate a good user interface, but not include any specific information about look or feel. Decisions about look and feel are left up to each interface generator. Included in the language are state variables and commands to represent the functions of the appliance, a hierarchical "group tree" to specify organization, dependency information that defines when states and commands are available to the user based on the values of other states, and multiple human-readable strings for each label in a specification.

The communication protocol allows user interface devices to download specifications, send control messages, and receive feedback messages that report the state of appliances. The two-way nature of this protocol allows the PUC to provide better user interfaces than an ordinary one-way remote control because of the feedback received.

One goal of the system is to control real appliances. Since there are no appliances available that natively implement the PUC protocol, translation layers must be built between the PUC protocol and the appliance's proprietary protocol. These translation layers are called appliance adaptors. A number of appliance adaptors have already been built, including a software adaptor for the AV/C protocol that can control most camcorders that support IEEE 1394 and another adaptor that controls Lutron lighting systems. Hardware adaptors have also been built for appliances that do not natively support any communication protocol. I am also interested in building general purpose adaptors to industry standards, such as UPnP and HAVi.

The last, but most important, piece of the PUC architecture is the interface generator. Interface generators have been built on several different platforms, including graphical interface generators on PocketPC, Microsoft's Smartphone, and desktop computers, and a speech interface generator that uses the Universal Speech Interfaces framework [12].

PROPOSED WORK

I have built a portion of the PUC system that is capable of generating the user interface for an appliance from an abstract specification [7]. Some automatically generated interfaces for Windows Media Player are shown in Figure 1. In this section I describe new pieces of the system that are currently being developed.

Domain-Specific Design Conventions

A common problem for automatic interface generators has been that their interface designs do not conform to domain-specific design conventions to which users are accustomed. For example, a good telephone user interface would include a standard number pad layout and a media player would use standard icons for play and stop. Solving this problem is particularly important for the PUC system because remote control interfaces should make use of design conventions.

I have developed one solution to this problem called *Smart Templates* [8]. A Smart Template is created for each situation where an interface generator might want to apply a design convention. Templates are standardized in advance so that specification designers know how to instantiate templates in an appliance specification and interface generators can include custom rules that recognize templates

[2] I am not addressing the device discovery problem in this system.

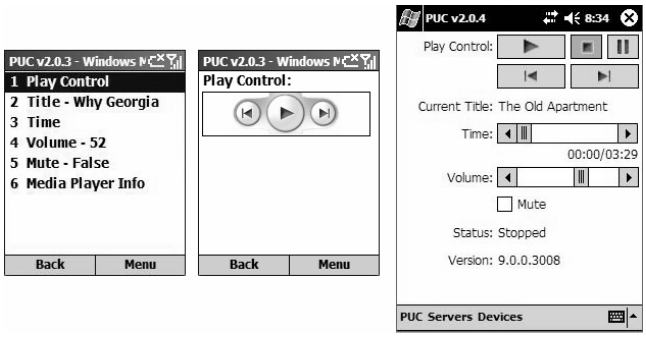

Figure 1. Automatically generated interfaces for Windows Media Player. On the left, two screens from a Smartphone interface. On the right, a screen from a PocketPC interface.

and render them appropriately. The custom rules ensure that the design convention is rendered in a small portion of an otherwise automatically generated interface.

For example, I have created a Smart Template for controlling media playback. This template allows interface generators to use the standard icons for play, stop, and pause, and to show each of these controls as buttons. Without the template, an interface generator would probably render these controls as a pull-down combo-box containing items for play, stop, and pause. This template is used in two of the interfaces shown in Figure 1. Note that a template may be rendered differently on different platforms.

I have built a preliminary implementation of Smart Templates into the PUC system, which supports some of the many Smart Templates that will be needed. I have developed a list of more than ten Smart Templates that I plan to implement, and I expect the list to grow as I look at new and different appliances. I also expect that some Smart Templates will naturally combine with others to create new templates. For example, date and time are often used together, as are volume and mute. I hope to implement Smart Templates so that templates can be flexibly combined with less work than creating a new template from scratch.

Interface Consistency

The PUC system has a unique opportunity to ensure external consistency among all interfaces that a user generates because PUC users have their own personal devices. This allows the PUC user interface device to remember previously generated interfaces and record usage statistics for those interfaces, which can be used to ensure that newly generated interfaces are consistent with older ones. For example, the interface that I generate for my new car stereo will probably be more usable if its layout is consistent with my home stereo interface that I use frequently.

The user interfaces generated by a PUC can be made consistent in two ways: 1) they can be consistent with other applications on the same controller device, and 2) they can be consistent with past interfaces generated for appliances with similar functions. The first can be achieved using the standard toolkit available on the controller device, and using generation rules that match the device's UI guidelines.

The second is more challenging and can be broken down into two sub-problems: finding previously generated interfaces that are relevant, and determining how to integrate decisions from the past interfaces into the new interface.

A relevant previously generated interface must include some of the same functionality as a new interface being generated. Unfortunately, it is difficult to conclusively know whether two functions on different appliances are the same. Functions may be similar if they have the same name, share some of the same labels, or have similar type information, but none of these alone conclusively show similarity. For example, a volume function might be represented by an integer ranging from 0-50 on one appliance and as an enumeration with 10 possible values on another appliance. I am exploring probabilistic solutions to this problem that can compare all similarity factors at once.

Once previous interfaces with similar functions have been found, the interface generator can examine those interfaces and decide how to make the new interface consistent. The appropriate technique will also depend on how similar the previous appliances are to the new appliance. So far I have identified three levels of similarity, termed sparse, branch, and significant (see Figure 2), each of which suggests a different technique to achieve consistency. Appliances with sparse similarity will try to represent each similar function with the same interface controls that the user saw in the older interface. Appliances with branch similarity will try

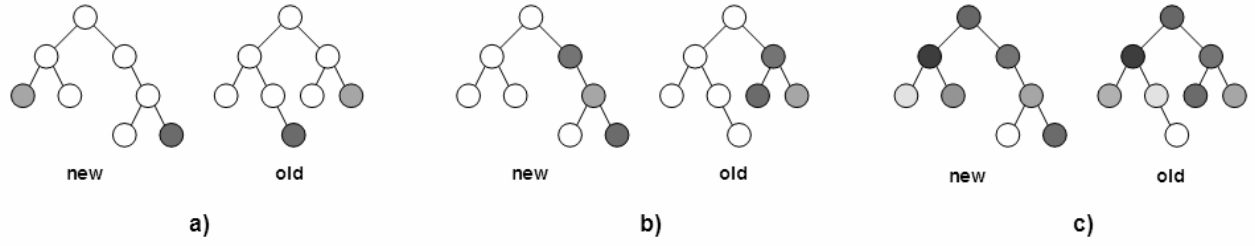

Figure 2. Examples of the three different levels of similarity. The trees represent the structure of the user interface as given by the appliance specification, with one tree representing the new specification and the other a previously generated specification. Nodes with the same shading indicate functions that were found to be similar across both appliances. a) *sparse similarity*: the appliances have a small number of similar functions spread throughout the tree. b) *branch similarity*: the appliances have a number of similar functions in one branch of the structure. c) *significant similarity*: the appliances share many similar functions, though they might be organized differently.

to integrate into the new interface the layout and organization of the related functions in the previous interface. Appliances with significant similarity will try to replicate the same layout and organization in the new interface that the user has seen in previous interfaces.

Multi-Appliance User Interfaces

A novel feature of the PUC system will be its ability to generate a single user interface for multiple appliances that have been connected together. One example is for a typical home theater, which includes separate VCR, DVD player, television, and stereo appliances, but might be more easily thought of as a single integrated appliance. A PUC interface for a home theater would ideally have features like a "Play DVD" button that would turn on the appropriate appliances, set the TV and stereo to the appropriate inputs, and then tell the DVD player to "Play."

A key question is how to model the connections between appliances and the interactions that users have which span appliances. Ideally a wiring diagram showing how each appliance physically connects to the others will be the only piece of system-specific modeling that is required. Tasks that users want to perform might be assembled from the wiring diagram and sub-tasks that are stored as part of each appliance's specification. I plan to develop a new *distributed* task modeling language, based on previous languages such as ConcurTaskTrees [10], to facilitate this process.

EVALUATION

There are two ways in which the PUC system must be evaluated in order to be judged a success: *breadth* of appliances supported by the specification language and interface generators, and *quality* of the generated interfaces compared to the manufacturers' interfaces on the actual appliances. For breadth I will compile a list of appliances that are interesting for their complexity or for a unique feature, write specifications for these appliances, and generate interfaces from these specifications on each interface generation platform. To show quality, I will conduct a user study comparing performance on the generated interfaces and the manufacturers' interfaces on the actual appliances.

I would also like to compare the PUC system to previous systems that have automatically generated interfaces. Unfortunately, many of these systems are a few years old and I am not yet sure how to perform such a comparison.

CONCLUSIONS

Most appliances on the market today are computerized, and within the next ten years most appliances will also feature networking technologies. Unfortunately, the user interfaces for most of these appliances are likely to be complicated and difficult to use. I am developing a system called the *personal universal controller* that will address this problem by moving the user interface from the appliance to an intermediary "UI device." The UI device can be easier to use because its interfaces use conventions users expect, create interfaces that are consistent with previous interfaces, and allow control of multiple appliances via a single interface.

ACKNOWLEDGMENTS

I would like to thank Brad Myers for advising me on this work. This work was funded in part by grants from NSF, Microsoft, General Motors, and the Pittsburgh Digital Greenhouse, and equipment grants from Mitsubishi Electric Research Laboratories, VividLogic, Lutron, and Lantronix. The National Science Foundation has funded this work through a Graduate Research Fellowship and under Grant No. IIS-0117658.

REFERENCES

1. Brouwer-Janse, M.D., Bennett, R.W., Endo, T., van Nes, F.L., Strubbe, H.J., and Gentner, D.R. "Interfaces for consumer products: "how to camouflage the computer?"" in *CHI'1992*. Monterey, CA: pp. 287-290.
2. HAVi, "Home Audio/Video Interoperability," 2003. http://www.havi.org.
3. Hodes, T.D., Katz, R.H., Servan-Schreiber, E., and Rowe, L. "Composable ad-hoc mobile services for universal interaction," in *Proceedings of the Third annual ACM/IEEE international Conference on Mobile computing and networking (ACM Mobicom'97)*. 1997. Budapest Hungary: pp. 1 - 12.
4. Kim, W.C. and Foley, J.D. "Providing High-level Control and Expert Assistance in the User Interface Presentation Design," in *Proceedings of INTERCHI'93* Amsterdam, The Netherlands: pp. 430-437.
5. Nichols, J., "Automatically Generating Interfaces for Appliances," in *Advances of Pervasive Computing*, H.H. A. Ferscha, G. Kotsis, Editor 2004, pp. 105-110.
6. Nichols, J., Myers, B.A. "Studying The Use Of Handhelds to Control Smart Appliances," in *23rd International Conference on Distributed Computing Systems Workshops (ICDCS '03)*. 2003. Providence, RI: pp. 274-279.
7. Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R., Pignol, M. "Generating Remote Control Interfaces for Complex Appliances," in *UIST 2002*. Paris, France: pp. 161-170.
8. Nichols, J., Myers, B.A., Litwack, K. "Improving Automatic Interface Generation with Smart Templates," in *Intelligent User Interfaces*. 2004. Funchal, Portugal: pp. 286-288.
9. Olsen Jr., D.R., Jefferies, S., Nielsen, T., Moyes, W., and Fredrickson, P. "Cross-modal Interaction using Xweb," in *Proceedings of UIST'00*. San Diego, CA: pp. 191-200.
10. Paterno, F., Mancini, C., Meniconi, S. "ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models," in *INTERACT*. 1997. Sydney, Australia: pp. 362-269.
11. Ponnekanti, S.R., Lee, B., Fox, A., Hanrahan, P., and T.Winograd. "ICrafter: A service framework for ubiquitous computing environments," in *UBICOMP 2001*. Atlanta, Georgia: pp. 56-75.
12. Rosenfeld, R., Olsen, D., Rudnicky, A., "Universal Speech Interfaces." *interactions: New Visions of Human-Computer Interaction*, 2001. **VIII**(6): pp. 34-44.
13. Szekely, P. "Retrospective and Challenges for Model-Based Interface Development," in *2nd International Workshop on Computer-Aided Design of User Interfaces*. 1996. Namur: Namur University Press. pp. 1-27.
14. UPnP, "Universal Plug and Play Forum," 2003. http://www.upnp.org.