AUTOMATICALLY GENERATING USER INTERFACES FOR APPLIANCES

Jeffrey Nichols[1]

Abstract

I am exploring the use of automatic user interface generation for improving the experiences that users have with computerized appliances in their pervasive computing environment. I am building the personal universal controller *(PUC) system, which will automatically generate graphical and speech user interfaces for an appliance on intermediary "user interface devices" such as personal digital assistants (PDAs) or mobile phones. Each interface is generated from an abstract specification of the appliance, written in a language I am designing, that describes the complete functionality of the appliance. The PUC system will also be able to generate a single combined user interface for multiple connected appliances, such as a home theater. Additionally, I am exploring how previous interface design decisions can be incorporated into future interface designs to achieve consistency for the user. I intend to evaluate the completed system by conducting user studies that compare the automatically generated appliance interfaces to the functionally identical manufacturers' user interfaces.*

1. Introduction

The Pervasive 2004 conference web site defines pervasive computing as a vision "…in which almost every object in our everyday environment will be equipped with embedded processors, wireless communication facilities and embedded software to perceive, perform and control a multitude of tasks and functions." User interfaces will be needed in order for us to harness the power of these objects, and these interfaces will be vastly more usable if they are personalized to each user and consistent with interfaces that user has seen in the past. I am proposing to build a general user interface framework for "smart appliances" will help ensure that these goals are met.

I call my approach the Personal Universal Controller (PUC) [5]. This system requires each "smart appliance" to have an abstract description of its functions. Whenever the user requires a user interface to an appliance, they request a device they are carrying to download the abstract description and automatically generate a user interface that can be used to control the appliance. Automatic generation allows the user interface to be generated in an appropriate modality, customized for the user's preferences, and consistent with other interfaces that the user has used for similar appliances. Two-way communication is maintained at all times between the user interface and the appliance, enabling the user to both send control messages and receive feedback on the state of the appliance.

The PUC system currently supports the automatic generation of graphical and speech interfaces for common appliances that are available today, such as stereos, copiers, elevators, and the non-driving

[1] Human-Computer Interaction Institute, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh PA, 15218 USA. jeffreyn@cs.cmu.edu, http://www.cs.cmu.edu/~jeffreyn/

functions of a car. Interfaces can be generated on multiple platforms: graphical interfaces can be created for PocketPC, Smartphone, and desktop computers, and speech interfaces can be created using the Universal Speech Interfaces framework [10]. I am not building interface generators for other modalities, but any modality should be possible given an appropriate user interface toolkit.

2. Related Work

Automatic user interface generation has been investigated by many researchers in the past [11]. 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 [3], and I am developing new techniques that increase the likelihood of high quality interfaces without requiring designer intervention.

Researchers have also examined how appliances can be remotely controlled. Systems such as the Universal Interactor [2] and ICrafter [9] investigated infrastructures for distributing appliance user interfaces to controller devices in the environment. Though both of these systems supported simple automatic generation of user interfaces, both preferred 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 [7] provided an infrastructure for interactive computing that was analogous to world-wide web. All user interfaces in Xweb were automatically generated from an abstract description of the interactive service being controlled. The PUC system extends the ideas of Xweb with more detail in the specification language, support for ensuring consistent user interfaces across similar appliances, and the ability to generate a single user interface for controlling multiple appliances.

3. PUC Architecture

The PUC is designed to allow users to control appliances in their environment through an intermediary user interface. The intermediary interface might be a graphical user interface on a handheld computer or a mobile phone, or it could be a speech interface that uses microphones in the room. When a user decides to control an appliance, their controller device would download from that appliance an abstract functional description and use that description to automatically generate an interface for controlling that appliance. 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 (see Figure 1). All of these pieces are described in more detail elsewhere [5]. Automatic generation of user interfaces is enabled by the specification language, which allows each appliance to describe its functions in an abstract way. Our goal in designing this language was to include enough information to generate a good user interface, but not any specific information about look or feel. Decisions about look and feel are left up to each interface generator. Included in our 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 appropriate string is chosen based upon the available space in the automatic layout.

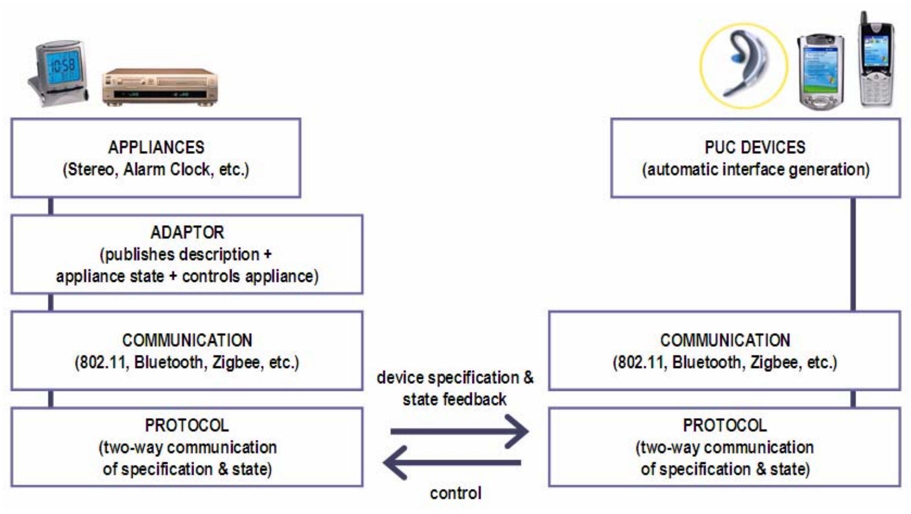

APPLIANCES
(Stereo, Alarm Clock, etc.)
ADAPTOR
(publishes description + appliance state + controls appliance)
COMMUNICATION
(802.11, Bluetooth, Zigbee, etc.)
PROTOCOL
(two-way communication of specification & state)
device specification & state feedback
control
PUC DEVICES
(automatic interface generation)
COMMUNICATION
(802.11, Bluetooth, Zigbee, etc.)
PROTOCOL
(two-way communication of specification & state)

Figure 1. An architectural diagram of the PUC system showing one connection (multiple connections are allowed at both ends).

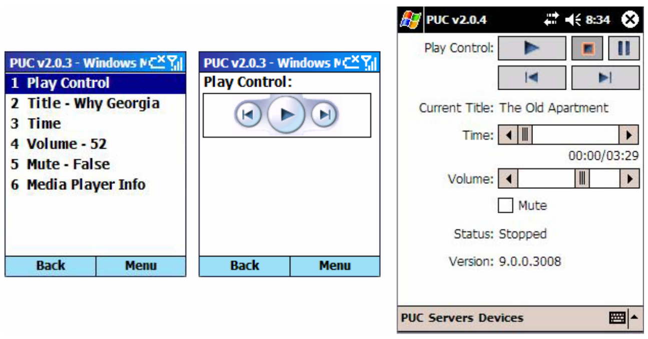
The communication protocol allows controller devices to download specifications, send control messages, and receive feedback messages that report the state of the appliance. The two-way nature of this protocol allows the PUC to provide a better user interface than an ordinary one-way remote control, because the user receives feedback on the success of his/her actions.

One goal of our system is to control real appliances. Since there are no appliances available that natively implement our protocol, we must build appliance adaptors, i.e. a translation layer between our protocol and an appliance’s proprietary protocol. We have built a number of appliance adaptors already, 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. We have also built hardware adaptors for appliances that do not natively support any communication protocol. We are also working on general purpose adaptors to industry standards such as HAVi [1] and UPnP [12]. This architecture allows a PUC to control virtually any appliance, provided the appliance has a built-in control protocol or someone has the hardware expertise to add one.

The last, but most important, piece of the PUC architecture is the interface generator. We have built interface generators 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 [10].

4. 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 [5]. Some automatically generated interfaces for Windows Media Player are shown in Figure 2. Here I describe new pieces of the system that I am proposing to build.



Current Title: The Old Apartment

Time: 00:00/03:29

Volume:

Status: Stopped

Version: 9.0.0.3008

Figure 2. Generated user interfaces for controlling Windows Media Player. On the left, two screens from a Smartphone interface. On the right, a screen from a PocketPC interface.

4.1. 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. Solving this problem is particularly important for the PUC system because remote control interfaces often make use of design conventions. For example, a telephone user interface should include a standard number pad layout and a media player should use the standard icons for the play and stop functions.

I have developed a solution to this problem called Smart Templates [6] that augments the PUC specification language’s primitive type information with high-level semantic information. If an interface generator understands a Smart Template then the appropriate design convention can be applied. If a Smart Template is not recognized, its functions can still be rendered because each template is based on the primitive elements of the specification language.

4.2. Interface Consistency

The user interfaces generated by a PUC can be made consistent for the user in two ways: 1) they can be consistent with other applications on the same controller device, and 2) they can be consistent with interfaces generated in the past for appliances with similar functions. The first can be achieved by 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. All previous interfaces will need to be saved, and usage statistics should be kept so that relevant interfaces can be ranked in order of user familiarity. Relevant interfaces might be found by searching for similar types or names in the appliance specifications, or by looking for occurrences of the same Smart Templates. One solution might be to fit the functions of the new appliance into the structure of a previous interface.

4.3. 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 of a use for this feature is 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 system. 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 that 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 [8], to facilitate this process.

5. Evaluation

I have two goals for the PUC system which must be evaluated in order to judge the system a success: breadth of appliances supported by the specification language and interface generators, and the quality of the generated interfaces as compared to the manufacturers' interfaces on the actual appliances.

I tried to choose my goals to be achievable but interesting. The goals are interesting because, as far as I know, no formal user-centric evaluation of a system for automatically generating interfaces has ever been conducted. The goals are achievable because though automatically generating high-quality interfaces from an abstract specification is a difficult task, the user interfaces created by most appliance manufacturers are not usually very easy-to-use. I also chose my goals to mirror the user studies that I conducted at the beginning of the PUC project, where I compared actual appliance interfaces with interfaces that I created by hand. The hand-designed interfaces showed a factor of two improvement in quality over the manufacturers' interfaces [4].

One issue with my quality goal is defining "usability." In previous user studies, I have analyzed the usability of interfaces by giving users a set of tasks to perform and measuring certain aspects of their performance while completing the tasks. In particular, I looked at the time to complete the tasks, the number of times a user manual was needed, and the number of missteps taken. This approach seems well suited to comparing user interfaces with identical functions, such as two user interfaces for the same appliance, and is the approach I intend to use for evaluating the PUC system.

Another problem is that it is impossible to prove that either goal has been achieved with absolute certainty. Doing so would require writing a specification for every appliance in existence, generating an interface from that specification, and then performing a comparative user study to show matching usability. I plan to deal with this issue by developing a list of appliances that are interesting, either because of their complexity or some unique feature, and testing them against my

Table 1. Appliances that the PUC system will be validated against.

Already Specified & Generated	Propose to Specify
Axis Pan & Tilt Camera	GMC Denali Navigation System
Audiophase Stereo	Phone/Answering Machine
GMC Denali Driver Information Center	Windows Media Player with playlists
GMC Denali Climate Control System	Automated Teller Machine (ATM)
Sony Camcorder	Microsoft Windows XP Media Center PC
Elevator Simulation	Photocopier
Lutron Home Lighting System	TV Tuner
Windows Media Player without playlists	Personal Video Recorder (e.g. TiVO)
X10 Lighting	Powerpoint
	Alarm Clock
	Projector

breadth goal. I will then choose a subset of those appliances, probably two or three, to test against my quality goal.

Table 1 shows the list of appliances that I have assembled so far, separated by those I have actually written specifications for. Note that a “GMC Denali” is a large sport utility vehicle manufactured by General Motors. I am particularly interested in feedback on this list, as I am sure there are other appliances I should be looking at.

6. Acknowledgements

I would like 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 funded this work through a Graduate Research Fellowship and under Grant No. IIS-0117658.

7. References

[1] HAVI, “Home Audio/Video Interoperability,” 2003. http://www.havi.org.

[2] 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.

[3] KIM, W.C. and FOLEY, J.D. “Providing High-level Control and Expert Assistance in the User Interface Presentation Design,” in *Proceedings INTERCHI'93: Human Factors in Computing Systems.* 1993. Amsterdam, The Netherlands: pp. 430-437.

[4] 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.

[5] NICHOLS, J., MYERS, B.A., HIGGINS, M., HUGHES, J., HARRIS, T.K., ROSENFELD, R., PIGNOL, M. “Generating Remote Control Interfaces for Complex Appliances,” in *Proceedings of UIST 2002.* Paris, France: pp. 161-170.

[6] NICHOLS, J., MYERS, B.A., LITWACK, K. “Improving Automatic Interface Generation with Smart Templates,” in *Intelligent User Interfaces.* 2004. Funchal, Portugal: pp. 286-288.

[7] OLSEN JR., D.R., JEFFERIES, S., NIELSEN, T., MOYES, W., and FREDRICKSON, P. “Cross-modal Interaction using Xweb,” in *Proceedings UIST'00: ACM SIGGRAPH Symposium on User Interface Software and Technology.* 2000. San Diego, CA: pp. 191-200.

[8] PATERNO, F., MANCINI, C., MENICONI, S. “ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models,” in *INTERACT.* 1997. Sydney, Australia: pp. 362-269.

[9] PONNEKANTI, S.R., LEE, B., FOX, A., HANRAHAN, P., and WINOGRAD, T. “ICrafter: A service framework for ubiquitous computing environments,” in *UBICOMP 2001.* 2001. Atlanta, Georgia: pp. 56-75.

[10] ROSENFELD, R., OLSEN, D., RUDNICKY, A., “Universal Speech Interfaces.” *interactions: New Visions of Human-Computer Interaction*, 2001. **VIII**(6): pp. 34-44.

[11] SZEKLEY, 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.

[12] UPNP, “Universal Plug and Play Forum,” 2003. http://www.upnp.org.