Automatically Generating Interfaces for Multi-Device Environments

Jeffrey Nichols and Brad A. Myers

Human Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
jeffreyn@cs.cmu.edu
http://www.cs.cmu.edu/~jeffreyn/

Abstract

With the increasing pervasiveness of wireless technologies, users will increasingly want to interact with appliances in their environment from their mobile device. One of the challenges to supporting such ubiquitous control is the creation of user interfaces for the mobile device. Should the mobile device be pre-programmed with interfaces for each appliance that it can control, should it download pre-designed user interfaces from each appliance, or should it download an abstract description of the appliance and then automatically generate a user interface to enable control? In this position paper we describe the Personal Universal Controller project, which is studying the automatic generation solution to this problem, and show how our infrastructure supports multi-device environments, which could contain a peripheral awareness device.

Introduction

Every day users interact with many computerized devices at both their home and office. On an average day I use my microwave oven, television, VCR, alarm clock, and stereo at home, and the copier, fax machine, telephone/answering machine, vending machine, and CD player at school. That does not count the mobile phone and wristwatch that I usually carry around with me, nor does it count the three "normal" computers that I use daily or many of the computerized components that my car would have if it had been built in the last ten years. All of these devices have different interfaces, even for functions that are common across most of them such as setting the internal clock. Even devices that are functionally similar, like my car stereo, home stereo, and the media player on my computer at school, have vastly different interfaces. As the user I have to spend time learning how to use every device in my environment, even when they are similar to other devices that I already know how to use.

One solution to this problem is move the interface to every appliance that I encounter to an intermediary "user interface" device. Such a UI device could make my life easier by creating similar user interfaces for appliances that have similar functions, or by combining the interfaces for a group of related appliances into a single interface. This would allow, for example, the interface I see for my car stereo to be similar to the interface for my home stereo. The interface for my home entertainment system, which consists of a TV, 2 VCRs, a DVD player, and a cable box, could also be combined into a single interface. This would be easier than finding and using four or five different remote controls as I am required to now.

It would even be possible for a user to have multiple UI devices, each with unique features. Some, like a mobile phone, PDA, and wristwatch, would be carried at all times but be used differently. The wristwatch could be the peripheral interaction device and be used only for fleeting or very casual encounters when little interaction is required. The mobile phone and/or PDA could be used for detailed interactions with complex appliances or data. In the home and office a user might have special UI devices built into the rooms that provide speech interfaces to appliances. The speech UI devices and the graphical UI devices, such as the PDA, could be used in tandem to make certain interactions more comfortable. The user might say "Play *Sweet Home Alabama*" to listen to a particular song, but use a slider on their PDA to set the volume rather than say "Volume: Louder Louder Louder".

There are three multi-device aspects to the architecture of such a UI device. The UI device and the appliance it is controlling together have a multi-device UI, since both have interfaces for controlling the appliance. The user might use the volume

slider on the UI device when they are sitting on the couch, but turn the knob on the actual stereo when they are standing next to it. Another multi-device aspect is the interaction of multiple UI devices each belonging to a different person. My sister and I might both sit on the couch with our UI devices and fight over the choice of which song the stereo will play next. Another example is that I might have some control over the thermostat in my office, but only to set the temperature between 60-90 degrees. A maintainance worker might be able to use his UI device to set the temperature arbitrarily or even turn off temperature regulation altogether. The third multi-device aspect is the ability of a single user to use more than one UI device at the same time. An example of this is the two UI devices used in tandem to control a stereo. A peripheral awareness device and a PDA is another combination of devices that might be used at the same time.

There are three different ways to build UI devices like these:

Pre-Programmed

The interface and the codes necessary to control a set of appliances are built into the UI device at the factory. This is similar to the way that current so-called "universal remote controls" work. It may be possible to download or "teach" the device additional codes after manufacture, but this process is often very tedious and time-consuming for the user. The Philips Pronto [Philips 2003] remote is one of the more advanced devices in this category. Its LCD panel and desktop computer sychronization features allow users to build custom control panels for each of their devices and to share their designs with other users. It is still tedious for users who buy a new appliance and find that they must program a new panel for their Pronto.

Downloadable Pre-Designed Interfaces

A UI device might connect directly to the appliance and download a user interface that can be used for control. Each appliance would come with several different interfaces that could be downloaded depending on the kind of device that wished to control it. For example, a photocopier might contain interfaces for both a PocketPC and a Nokia mobile phone. This model is used by the Java JINI system as one mechanism for distributing user interfaces. The problem with this approach is that the appliance must either contain pre-designed interfaces for every device that can control it, or the user must be forced to use interfaces not explicitly designed for their device. A user with a Palm might be forced to use an interface that was designed for a PocketPC, for example. Another problem is that the appliance manufacturer cannot anticipate all devices that will be created in the future. Next year, for example, some manufacturer might come out with a computerized watch that uses a dial and four buttons as the only methods of interaction. An appliance built this year would probably not contain an interface that would work with this new device. This is an especially large problem when you consider that appliances are expected to last many years, whereas new handheld devices emerge about every six months.

Automatic Interface Generation

A UI device might connect directly to the appliance and download from it an abstract description of its functions. This description would not contain any layout information or any description of the types of controls to be used in the interface. From the abstract description, the UI device would *automatically* generate an interface that the user could use to control the appliance. Such a system would be able to create interfaces that are consistent with other applications on the same device (e.g. interfaces generated for a PocketPC would use the same conventions as other PocketPC applications). Interfaces could also be generated based on past generated interfaces, allowing the interface for my car stereo to look and feel similar to the interface for my home stereo.

We have a built a system that we call the Personal Universal Controller (PUC), which uses automatic interface generation to allow users to control appliances in their environment [Nichols 2002b]. As a part of this work, we have designed an abstract specification language for describing the features of an appliance. We have also created interface generators that use this language to build interfaces for Microsoft's PocketPC, Smartphone and TabletPC, each of which have different interaction styles, and also to build speech interfaces in the Universal Speech Interfaces framework [Rosenfeld 2001]. Figure 1, Figure 2, and Figure 3 show interfaces generated for Windows Media Player on a PocketPC, Smartphone, and desktop respectively.

Figure 1. Interface for Windows Media Player automatically generated for a PocketPC.

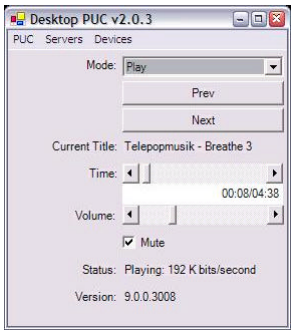

Figure 3. Interface for Windows Media Player automatically generated for a desktop computer.

Figure 2. Interface for Windows Media Player automatically generated for a Microsoft Smartphone

Personal Universal Controller

Our approach to this work has differed from previous work. Rather than start by trying to abstractly describe appliances, we started by hand-designing remote control interfaces on the Palm and PocketPC platforms for two different appliances: an Aiwa CX-NMT70 shelf stereo and the AT&T 1825 telephone/answering machine. In a comparison of our hand-designed interfaces to the manufacturer's interfaces on the actual appliances, we found that users were *twice as fast* and made *half as many errors* using our hand-designed interfaces [Nichols 2003]. By analyzing the hand-designed interfaces we derived a list of requirements for systems that automatically generate interfaces [Nichols 2002a], and determined what information should be included in our abstract specification language.

Our specification language contains the following types of information:

State Variables and Commands

The functions of an appliance are represented in our language with state variables and commands, as other systems have done [UPnP 2003]. State variables have types, such as integer, boolean, or enumerated, and these types are used to choose the controls that will represent the variables in the generated user interface.

Multiple Labels

In our specification language designers may specify more than one text string for each label in blocks that we call *label dictionaries*. For example, the Power state variable might use: "Power", "Powr", and "Pwr".

Hierarchical Grouping

State variables and commands are arranged into a tree that interface generators can use to determine structure. These trees are purposely made deeper rather than wide so that more structural information is available for interface generators for small screens that can only show a few controls at one time.

Dependency Information

A novel feature of our specification language is dependency information, which defines when a particular state or command is active, based on the values of other state variables. Our system supports relations such as `equals`, `greater-than`, and `less-than` which can be grouped with the logical operators `AND` and `OR`. Dependency information is used to "gray-out" controls that are linked with state variables that are not active, and it can also be useful for determining layout. For example, if there are two groups of controls that are never available at the same time then an interface generator can place these groups on overlapping panels.

Our specification language is based on the XML standard and is fully documented. For more information on the language, see http://www.cs.cmu.edu/~pebbles/puc/specification.html.

We have built several interface generators that use our abstract specification language to automatically generate interfaces. We have built generators for the Microsoft PocketPC, the Microsoft Smartphone, and for desktop computers (See Figures 1, 2, and 3 respectively). All of these generators are implemented using C# and the .NET Compact Framework, and each uses a rule-based approach to generate interfaces. Currently each interface is generated only from the specification language, but in the future we expect to add other models into the process, such as a model of the user. This would allow the interface generators to incorporate user preferences into the automatic designs.

A PUC interface generator for speech interfaces has also been created by the Universal Speech Interfaces (USI) group [Rosenfeld 2003] at Carnegie Mellon. The interfaces generated by this system allow the user to navigate around in a virtual tree and issue commands based on their current location in the hierarchy. Dependency information is used to disambiguate commands where possible. For example, if the user said "Play" the system would determine that it should play the CD and not the tape because the tape player is not active. Dependencies are also used by the system if the user explicitly requests a feature that is inactive. For example, if the user said "Play Tape" then the system would change the stereo mode to "Tape" (because this is a dependency for the tape player) and then start the tape playing.

Multi-Device User Interfaces and the PUC

Using the PUC framework, it is possible to generate an interface for controlling a particular appliance on more than one device. A user could generate a speech interface and a PocketPC interface for the same appliance, and pick and choose which interface to use based upon the current situation. The user might choose to use only the speech interface in a car while driving for example, but only use the PocketPC interface when sitting at home watching a movie with the family.

The PUC has built-in support for all three aspects of multi-device computing mentioned earlier. A user can control other devices with their controller, multiple users can control the same device, each using their own controller, and one user can use multiple controllers simultaneously to control any number of appliances. All controllers are notified when any appliance states change, but the PUC framework does not have any other coordinated functions for the third aspect that might make it easier for one user to use multiple controller devices. Every controller in a PUC system has an interface for exactly the same set of appliance functionality and no explicit knowledge of when or how a user is interacting with another controller. Controllers do know when the state of an appliance changes, but they do not know if these changes were caused by a user or by normal operation of the appliance. The counter on a VCR changes regularly when a tape is playing, for example. To make a truly good multi-device interface for a single user, it may be necessary for controllers to share information about themselves. This would allow controllers to do at least two things:

Support A Fluid User Experience

If a user is controlling the lights in their house from the speech interface and then suddenly wants to continue the interaction on a PocketPC interface, it would be helpful if the PocketPC controller had automatically switched to the lighting interface panel.

Specialize

Current controllers all support the same set of functionality. In a multi-device environment, users might prefer each of their controllers to specialize, and offer access to a subset of functions that make the most sense for the modality and form factor of the controller. When the set of controllers in an environment changes, the remaining set can alter their functionality appropriately.

The idea of specialization is particularly important for impoverished devices such as a ubiquitous peripheral device that has only a few buttons and an extremely small screen. An interface generated for such a device should probably only present a subset of the most commonly used features by default. Another option is to allow the user easy access to common features

and make the less common features available but not very accessible, perhaps in a long scrolling list. It is important that the peripheral device allow for some control however, and not be purely an information notification device. It may be time-consuming for the user to get out their PDA every time they need to perform a quick interaction, whereas the peripheral device is always close at hand.

Conclusion

The Personal Universal Controller (PUC) system allows users to control appliances using the devices that they carry with them. This system creates interfaces automatically from an abstract specification language, rather than by downloading a pre-designed interface or by being pre-programmed with a set of interfaces from the start. The automatic generation feature gives the system flexibility to create interfaces that are consistent with the device they are running on and to incorporate user preferences. With automatic generators running on a number of different devices, it is possible to create a multi-device interface to an appliance where the user can flexibly change controllers to suit the interaction. We believe the PUC system is a good basis for a multi-device interface system, though some changes might be necessary to improve compatibility with a peripheral awareness device.

Other Relevant Pebbles Work

The Personal Universal Controller is a part of the Pebbles project [Myers 2001], the goal of which is to explore how handheld and desktop computers can be used together simultaneously. Remote Commander, the first Pebbles application, allows a handheld to be used as the mouse and keyboard for a desktop. This application was targeted at meetings to allow every participant to control a single desktop computer without leaving their seats. The Slideshow Commander (SSC) application allows users to control Powerpoint presentations from their handhelds, which can also be useful in meetings. Multiple users running SSC could write on the screen simultaneously, possibly facilitating discussions that occur as a part of the presentation. The Shortcutter application allows users to create custom panels that control various aspects of their desktop computer. A user might make a custom panel with two scrollbars that are linked to the horizontal and vertical scrollbars that appear in many applications. A common use of Shortcutter is to control an MP3 playing application on a computer that is not easily accessible. Each of these applications, and the many others developed as part of the Pebbles Project, explore how a handheld device can be part of a multi-device UI by pushing pieces of desktop computer user interfaces onto the handheld.

References

[Myers 2001] Myers, B.A., "Using Hand-Held Devices and PCs Together." *Communications of the ACM*, 2001. **44**(11): pp. 34-41.

[Nichols 2002a] Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R., Shriver, S. "Requirements for Automatically Generating Multi-Modal Interfaces for Complex Appliances," in *Proceedings of ICMI 2002*. Pittsburgh, PA: Oct. 14-16. pp. 377-382

[Nichols 2002b] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld, Mathilde Pignol. "Generating Remote Control Interfaces for Complex Appliances," In *Proceedings of UIST'2002*, Paris, France. Oct 27-30. pp. 161-170

[Nichols 2003] Jeffrey Nichols and Brad A. Myers. "Studying The Use Of Handhelds to Control Smart Appliances," In *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCS '03)*. Providence, RI. May 19-22, 2003. pp. 274-279

[Philips 2003] Philips, *Pronto Intelligent Remote Control*. Philips Consumer Electronics, 2003. http://www.pronto.philips.com/

[Rosenfeld 2001] Rosenfeld, R., Olsen, D., Rudnicky, A., "Universal Speech Interfaces." *interactions: New Visions of Human-Computer Interaction*, 2001. **VIII**(6): pp. 34-44.

[Rosenfeld 2003] Rosenfeld, R., "Universal Speech Interfaces Web Site," 2003. http://www.cs.cmu.edu/~usi/.

[UPnP 2003] UPnP Forum, "Universal Plug and Play Forum," 2003. http://www.upnp.org.

Acknowledgements

This work was conducted as a part of the Pebbles [Myers 2001] project, with the help of many other people including Michael Higgins and Joseph Hughes of MAYA Design, Thomas K. Harris, Roni Rosenfeld, Kevin Litwack, Mathilde Pignol, Rajesh Seenichamy, and Stefanie Shriver of Carnegie Mellon University. The speech interface was implemented as a part of the Universal Speech Interfaces project [Rosenfeld 2003]. This work was funded in part by grants from NSF, Microsoft, General Motors, DARPA, and the Pittsburgh Digital Greenhouse, and equipment grants from Mitsubishi Electric Research Laboratories, VividLogic, Lutron, Lantronix, IBM Canada, Symbol Technologies, Hewlett-Packard, and Lucent. The National Science Foundation funded this work through a Graduate Research Fellowship for the first author and under Grant No. IIS-0117658. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.