Improving Automatic Interface Generation with Smart Templates

Jeffrey Nichols, Brad A. Myers, and Kevin Litwack
Human Computer Interaction Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
http://www.cs.cmu.edu/~pebbles/puc/

{jeffreyn, bam, klitwack}@cs.cmu.edu

ABSTRACT

One of the challenges of using mobile devices for ubiquitous remote control is the creation of the user interface. If automatically generated designs are used, then they must be close in quality to hand-designed interfaces. Automatically generated interfaces can be dramatically improved if they use standard conventions to which users are accustomed, such as the arrangement of buttons on a telephone dial-pad or the conventional play, stop, and pause icons on a media player. Unfortunately, it can be difficult for a system to determine where to apply design conventions because each appliance may represent its functionality differently. *Smart Templates* is a technique that uses parameterized templates in the appliance model to specify when such conventions might be automatically applied in the user interface. Our templates easily adapt to existing appliance models and interface generators on different platforms can apply appropriate design conventions using templates.

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.*

General Terms

Algorithms, Design, Human Factors, Languages

Keywords

Automatic interface generation, Pebbles, handheld computers, appliances, personal digital assistants (PDAs), personal universal controller (PUC)

1. INTRODUCTION

A common problem for automatic interface generators has been that their interface designs do not conform to domain-specific design patterns that users are accustomed to. For example, an automated tool is unlikely to produce a standard telephone keypad layout. This problem is challenging for two reasons: the user interface conventions used by designers must be described, and the interface generators must be able to recognize where to apply the conventions through analysis of the interface specification. Some systems [7] have dealt with this problem by defining specific rules for each application that apply the appropriate design conventions. Other systems [2] rely on human designers to add design conventions to the interfaces after they are automatically generated. We have built the first system we are aware of that applies design conventions automatically for a wide variety of applications without defining specific rules for each generated interface.

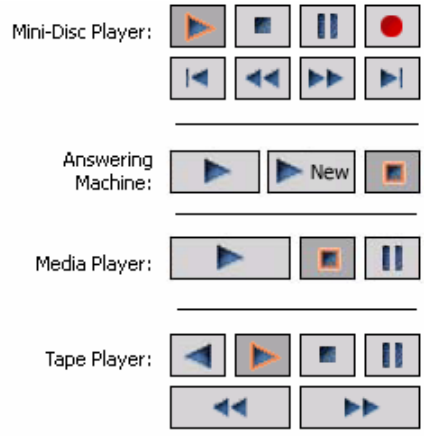

Figure 1. Arrangements of media playback controls generated automatically from the `media-control` Smart Template.

We are building a system called the *personal universal controller* (PUC) that automatically generates user interfaces for remotely controlling all the appliances in a user's environment [3]. Unlike most previous model-based work, user interfaces generated by a PUC are intended for immediate use by people who are not trained interface designers. Therefore the user interfaces must be complete and usable without further modification. We do not expect that a user will want to walk up to their office photocopier with an important document, pull out their PUC device, and then spend time fixing the layout of their copier user interface. We have created PUC interface generators for multiple platforms, including PocketPC, Microsoft's Smartphone, desktop computers, and speech interfaces.

We have recently added a feature to the PUC system called *Smart Templates*, which extend the primitive types in the PUC appliance

IUI'04, Jan. 13–16, 2004, Madeira, Funchal, Portugal.
ACM 1-58113-815-6/04/0001.

specification language with high-level semantic information. For example, the `media-controls` template defines that a state variable with an enumerated type containing members labeled *Play* and *Stop* controls the playback of some media. PUC interface generators can use the information added by a Smart Template to apply design conventions and make interfaces more usable. If an interface generator does not recognize a template however, a user interface can still be created because Smart Templates are constructed from the primitive elements of our specification language.

An important innovation is that Smart Templates are parameterized, which allows them to cover both the common and unique functions of an appliance. For example, the media playback template supports play and stop, but also optional related functions such as next track for CDs, fast-forward and reverse-play for tape players, and "play new" for phone answering machines (see Figure 1). Smart Templates also give appliances the flexibility to choose a functional representation that matches their internal implementation. For example, our `time-duration` Smart Template allows single state variables with integer or string types, or multiple state variables (e.g. a state for hours and another for minutes).

The PUC specification language and interface generators have been described elsewhere [3]. That paper mentions the need to describe high-level semantic information in our specification language, but does not offer a solution. This paper is the first discussion of how the PUC incorporates high-level semantic information.

2. RELATED WORK

Earlier model-based systems had some support for using design conventions in their generated interfaces. ITS [7] allows the specification of style rules, which could be shared across multiple applications. A style rule could create a conventional layout if it found particular relationships in ITS's action or dialog layers. In practice, however, it was found that it was difficult to write and share style rules. HUMANOID [6] has the ability to apply custom displays to application-specific types. These displays are specific to the application however, and it is not clear whether they can be easily shared among applications. The display would also need to be modified if the representation of the type changed between applications. Most other model-based systems offer interface designers the ability to modify the interfaces after they were generated [2]. An important principle of Smart Templates is that no modifications are necessary since they allow conventional layouts to be applied automatically during interface generation.

A number of systems exist for controlling appliances, many of which support the automatic generation of user interfaces [1, 5]. Of these, only the Ubiquitous Interactor (UBI) [4] has a technique for incorporating design conventions, although it is unclear whether UBI's customization forms are portable between different appliances. The Smart Template technique that we present here is general, and could probably be added to all of these systems.

3. SMART TEMPLATES

A Smart Template augments the grouping and primitive type information in our appliance specification language with knowledge about the semantics of the data. Smart Templates are *defined* in advance by template writers, who specify the set of states and commands that can be included in a template. Some items are required and others are optional to allow flexibility in the different functions an appliance may support and their implementations. Smart Templates are *used* in the appliance specification to add semantic information to either a group or state variable with the special `is-a` attribute (see Figure 2). Adding this attribute requires the appliance specification to conform to the definition of that Smart Template. Finally, Smart Templates are *rendered* by interface generators based upon the chosen template and the content that was included within the template in the appliance specification.

```
<group name="Counter" is-a="time-duration">
  <labels> <label>Counter</label> </labels>

  <state name="Hours">
    <type>
      <integer/>
    </type>
  </state>

  <state name="Minutes">
    <type>
      <integer>
        <min>0</min> <max>59</max>
      </integer>
    </type>
  </state>
</group>
```

```
<state name="SongLength" is-a="time-duration">
  <type>
    <string/>
  </type>

  <labels> <label>Length</label> </labels>
</state>
```

Figure 2. Two examples of `time-duration` Smart Template instances from PUC appliance specifications.

3.1 Definition

A template writer starts by selecting the state variables and commands that the template may contain, and then defines the names, types, values, and other properties that these elements must have. The challenge for the template writer is to find the different combinations of states and commands that an appliance implementer is likely to use. This makes it easier for appliance specification writers to use the templates, because there is no need to modify the appliance's internal data representation in order to interface with the controller infrastructure. For example, Windows Media Player makes the duration of a song available as a single integer while our Sony DV Camcorder makes the playback counter available as a string. Using our `time-duration` Smart Template, both of these representations can be handled appropriately by an interface generator, while still providing a consistent look and feel to users.

Allowing many combinations of states and commands in a template definition also allows a single Smart Template to be applied across multiple kinds of appliances. Two representations of play controls are allowed by the `media-controls` template: a single state with an enumerated type, or a set of commands. If a single state is used, then each item of the enumeration must be labeled. Some labels must be used, such as Play and Stop, and others are optional, such as Record. If multiple commands are used, then

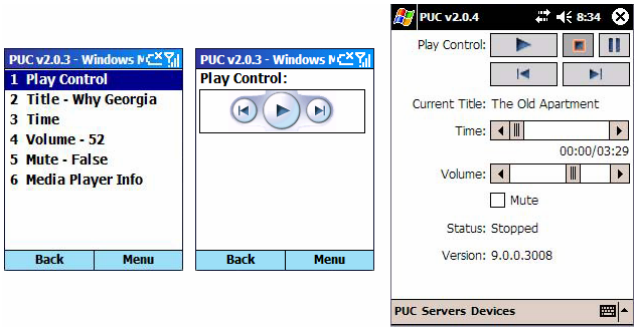

Figure 3. Generated user interfaces for controlling Windows Media Player. On the left, two screens from a Smartphone interface. On the right, a screen from a PocketPC interface.

each command must represent a function such as Play and Stop. Some functions must be represented by a command and others are optional. This template also allows three commands for functions that are commonly included in the same group as the play controls, including the previous and next track functions for CD and MP3 players, and the play new function for answering machines.

We have defined several templates and plan to define many more.

3.2 Rendering

Smart Templates allow interface generators to use platform-specific controls that are consistent with other user interfaces on the same device. In the case of our `time-duration` Smart Template implementation, each platform has a different standard control for manipulating time that our interface generators use. Unfortunately, none of our platforms have built-in controls for media playback, so our `media-controls` Smart Template does not benefit in the same way. The Smartphone `media-controls` implementation does mimic the interface used by the Smartphone version of Windows Media Player however, and thus is consistent with another application on that device (see Figure 3).

Smart Templates are also able to intelligently choose a rendering based upon the contents of the template. For example, each implementation of the `time-duration` template only renders the time units that are meaningful, and each implementation of our `media-controls` Smart Template renders buttons for only the functions that are available. The `media-controls` implementations on the PocketPC and desktop extend this by intelligently laying out the buttons on one or more lines depending on space, enlarging buttons of greater importance such as Play, and using a grid to create aesthetically pleasing arrangements (see Figure 1).

4. FUTURE WORK

In the immediate future we plan to define and implement many new Smart Templates in our system. As we connect our system to more appliances, we expect that we will find many more uses for Smart Templates. A goal of our work is to create a comprehensive list of templates for common appliances available for others.

We also plan to implement Smart Templates in our speech interface generator. This should improve certain aspects of those interfaces, especially for common types such as date and time. The current speech generator does not speak a time properly (saying *one colon two four*, for 1:24) because it does not know that the value represents a time. With a Smart Template, the speech system will be able to use the extra semantic information to improve its speaking ability. We believe there are many other situations where semantic information can benefit our speech interface generator.

Finally, we intend to conduct user studies to compare the interfaces generated by a PUC with the interfaces developed by manufacturers' for their own appliances. Our goal is to show that users' performance with our generated interfaces match or exceed their performance with the manufacturers' interfaces.

5. CONCLUSION

We have described Smart Templates, a technique for improving automatically generated interfaces. This technique is novel because it uses parameterized templates to allow automatic interface generators to create interfaces that are consistent and more usable. Parameterization makes appliance specifications easier to create because it does not require appliances to be implemented in a particular way. Smart Templates can be used by interface generators to render basic elements such as time and more complex structures such as the playback controls for a media player. If an interface generator does not recognize a template, it can still be rendered because templates are described in terms of the primitive elements of our description language. Automatic interface generators can use Smart Templates to improve their interfaces by using layouts that are consistent with other interfaces on the same device and with other appliances in the world.

6. ACKNOWLEDGEMENTS

This work was conducted as a part of the Pebbles project. The speech interface was implemented as a part of the Universal Speech Interfaces project. 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, and Lantronix. 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.

7. REFERENCES

[1] Gajos, K., Weld, D. "SUPPLE: Automatically Generating User Interfaces," in *Intelligent User Interfaces.* 2004. Funchal, Portugal:

[2] 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.

[3] 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.

[4] Nylander, S. "Different Approaches to Achieving Device Independence," in *Technical Report TR2003-XX.* 2003. Swedish Institute of Computer Science:

[5] 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.

[6] Szekely, P., Luo, P., and Neches, R. "Beyond Interface Builders: Model-Based Interface Tools," in *Proceedings INTERCHI'93: Human Factors in Computing Systems.* 1993. Amsterdam, The Netherlands: pp. 383-390.

[7] Wiecha, C., Bennett, W., Boies, S., Gould, J., and Greene, S., "ITS: A Tool for Rapidly Developing Interactive Applications." *ACM Transactions on Information Systems*, 1990. **8**(3): pp. 204-236.