Directions in Ubiquitous Computing: A Position Statement
Olufisayo Omojokun & Prasun Dewan
University of North Carolina at Chapel Hill
{ojojokun, dewan} @cs.unc.edu

1. Interacting with Networked Appliances

One of the visions of ubiquitous computing is the ability to use arbitrary interactive devices such as cell phones and handheld computers to interact with arbitrary remote networked appliances such as TVs, printers, and EKG machines. There are many potential practical benefits of controlling a network appliance using a remote user-interface on a mobile computer rather than directly using the native, physical user-interface offered by it. Users can control appliances from arbitrary locations – for instance, people on vacation can set their home water sprinklers. Moreover, an interactive mobile device can be used as a true universal control, accessing arbitrary devices such as TVs, thermostats, and light switches. Furthermore, a mobile device can offer user-interfaces that are more sophisticated than the physical user-interfaces offered by the appliances directly. For instance, it can offer a single command to shut off all lights in a building.

2. The Deployment Issue

One of the questions raised by this vision is: How should a user-interface agent be deployed on a mobile interactive computer to interact with a remote networked appliance? The most straightforward approach is to load it from local storage on the mobile computer. This is the approach supported by commercial applications that allow Palm devices to control VCRs, TVs, CD players, and stereo systems through infrared. Unfortunately, it does not allow the mobile device to interact with an appliance it has never seen before. A more flexible approach is to fetch the user-interface agent from some remote computer, which could be the appliance or some third-party server. This approach also has problems such as the security risks and time costs of downloading code, and the need to somehow describe to the remote site the characteristics of the interactive device so that an appropriate user-interface agent for it can be fetched - currently an unsolved problem. We refer to these two approaches as predefined approaches, because they place pre-existing user-interface agents at well-known locations for the deployers to find. A more flexible approach is to generate a device-specific user-interface agent ‘on the fly’ based on the functionality of the appliance. Previous work [1] suggests the value of automatic user-interface generation in many domains, and recently its role in ubiquitous computing been suggested by an ongoing Berkeley project [2].

3. User-Interface Generation

The Berkeley system illustrates how user-interface generation might be done. It requires that an XML document reveal to the generator the operations provided by the appliance. This document is independent of the particular language in which the operations are implemented—it uses a standard schema for describing the operations. Thus it offers the flexibility of using any language to code the appliance. Figure 1 shows an outline of Java code implementing the operations of a lamp.

```java
public interface Lamp {
    public void dim_halfway();
    public void strobe();
    ... }
```

![Figure 1. Lamp code.](image1)

An example XML description for the lamp is given in Figure 2. As the figure shows, the XML approach defines several tags, which provide heuristics to the generator for creating the user interface. Figure 3 illustrates a possible user-interface generated from the XML document above.

![Figure 2. Lamp XML document](image2)

![Figure 3. Lamp UI.](image3)

4. Position Statement

User-interface generation is a promising research direction in ubiquitous computing because it allows a mobile device to interact with an arbitrary appliance without incurring the problems of downloading and classifying user-interface code. On the other hand, the generation approach offers limited presentation styles. We
believe this is not a serious drawback, for three reasons: First, because of hardware limitations of mobile interactive devices, there is not much opportunity for variability in the kinds of presentation styles and other user-interface features they can present. For instance, we expect the variations in the kinds of presentation styles a four-line text-only display cell-phone can create to be much less than that of a full-fledged PC. Secondly, the presentation styles and commands can be as simple as those provided by traditional remote controls. Third, as the number of networked appliances mobile users interact with gets large, it will not be possible for them to learn appliance-specific user-interface features, and they will thus demand a standard set of features. Nonetheless, the generation approach supports a subset of possible presentation styles, and does not support some features important in the domain of networked appliances, such as multi-appliance control.

The Berkeley project, however, is only an initial step in user-interface generation, and with more research it can become a practical idea. In the rest of the paper, we first describe the steps we have taken so far to advance this idea, and then discuss other steps that we plan to take.

5. Our Approach

Instead of requiring the appliance developer to provide two descriptions of the operations of an appliance, one in the language in which the appliance is coded (Figure 1) and the other in XML (Figure 2), we require them to provide only the former. The user-interface generator uses the programming interface rather than the XML description to create the user-interface. Thus, our system can be used to generate a user-interface similar to the one created by the Berkeley system.

The Berkeley approach has the advantage that it is language-neutral, giving the programmer the flexibility of choosing arbitrary languages. On the other hand, in comparison to our language-based approach, it has two drawbacks. First, it offers higher programming and maintenance costs. In current systems, the external description is manually specified and changed in response to changes to application functionality. Moreover, the appliance programmer is responsible for writing and maintaining code that convert between language-independent and language-based operations. The code required to perform these tasks can amount to a significant portion of the total appliance code. Second, it offers higher operation invocation and event propagation cost because of the cost of translating between language-dependent and language-independent values. There is anecdotal evidence to show that this cost can be significant. Thus, both approaches have important advantages. Another contribution of our work is to show that user-interface generation can be used to create one user-interface for interacting with a set of appliances. The user-interface shows the union of the operations offered by the appliances in the set, much as a universal remote control does (Figure 4). Unlike the latter, however, our approach gives the user the flexibility to execute an operation simultaneously on all or a selected set of appliances. The predefined approach cannot support multi-appliance interaction. The reason is that a predefined user-interface is bound to a preexisting set of appliances whereas multi-appliance interaction allows this set to be dynamic, such as all appliances in a house. The generation approach is consistent with multi-appliance interaction because it dynamically generates the user-interface—instead of taking a single appliance description as input, it can process a set of appliance descriptions.

6. Other Directions

Based on this discussion, several future research directions can be identified. It is necessary to carefully identify the advantages (such as multi-appliance interaction) that mobile computers offer over traditional remote controls. A more careful evaluation of the various forms of predefined and generation approaches is necessary, which in turn requires identification of criteria (such as programming and maintenance cost, flexibility, and deployment time mentioned here) for evaluating them. Finally, new approaches need to be developed that combine the benefits of existing approaches. For example, it would be useful to automatically generate the language-neutral XML description from the appliance code—thereby offering the programming and maintenance costs (but not the efficiency) of the language-based approach in the language-neutral approach.

8. Acknowledgments

This work was supported in part by U.S. National Science Foundation Grant Nos. CDA-9624662 and IIS-997362.

9. References
