Design Patterns for Ubiquitous Computing

James A. Landay, University of California, Berkeley
Gaetano Borriello, University of Washington

By communicating solutions to common problems, design patterns make it easier to focus efforts on unique issues.

Design is about finding solutions. Unfortunately, not knowing how others previously applied a solution or why they did things a certain way makes it difficult to reuse prior design knowledge. Consequently, designers—whether they are software engineers, architects, or Web designers—often end up having to reinvent the wheel.

Because they are neither too general nor too specific, *design patterns* offer a solution to the difficult problem of reusing prior design knowledge. They are written to be flexible enough for reuse in many situations, and designers can use them to identify and propose solutions to recurring problems. We propose that such patterns also offer an effective way to communicate solutions to ubiquitous computing design problems.

EVOLUTION OF DESIGN PATTERNS

Design patterns emerged from the field of architecture in the 1970s with the work of Christopher Alexander and his colleagues at the University of California, Berkeley. In the still widely read classic *A Pattern Language: Towns, Building, Construction* (Oxford Univ. Press, 1977), they proposed a set of 253 formal patterns not only as a tool for architects, but as a way for all stakeholders in the design process, including the client, to communicate about a given project.

Sample architectural design pattern

The Beer Hall pattern provides one example of Alexander's distinctive approach to design. This pattern addresses the following problem:

> Where can people sing and drink, and shout and drink, and let go of their sorrows?

The solution is:

> Somewhere in the community at least one big place where a few hundred people can gather, with beer and wine, music, and perhaps a half-dozen activities, so that people are continuously crisscrossing from one to another.

Figure 1 shows an abstract sketch of this solution. To accommodate such a large gathering of people, a beer hall should have tables in the middle and activities around the circumference to encourage movement throughout the room.

The description of a pattern is typically three to five pages long and discusses forces, or tradeoffs, that designers must consider to successfully apply the pattern. For example, in designing a beer hall: What should be the focus of attention? How will people mingle? How will they feel they are in a place of their own?

Design patterns range in scale from a city to a room and, together, form a *pattern language* that designers can adapt to a project's particular level of complexity or detail. For example, in designing a beer hall, you could select Alcoves, a subpattern that states that any common room should have small spaces at the edge "large enough for two people

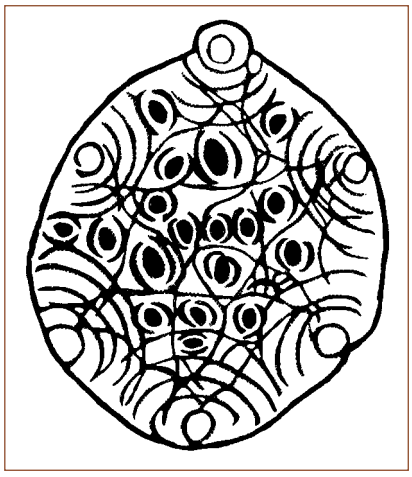

Figure 1. Beer Hall design pattern. A beer hall should have tables in the middle and activities around the circumference.

Figure 2. Context-Sensitive I/O design pattern. A cell phone should detect whether the owner is driving or at a meeting and accordingly either ring or vibrate.

to sit, chat, or play and sometimes large enough to contain a desk or a table." Likewise, Beer Hall could be an integral pattern in Night Life, a higher-order pattern that recommends knitting together "shops, amusements, and services which are open at night, along with hotels, bars, and all-night diners to form centers of night life."

Software design

Alexander's ideas have caught on in disciplines other than architecture. Since the mid-1990s, design patterns have been one of the most influential ideas in software engineering, first popularized by the "gang of four"—Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides—in their book, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, 1995). For example, the Listener pattern offers a solution to the problem of how to notify one part of a program of an event that has occurred elsewhere in the program. Software design patterns gave concrete names to abstract concepts that many developers had "discovered" themselves, making such concepts easy to share.

UI design

More recently, design patterns have sparked interest in user interface design, a field that shares the customer focus of architecture. Jenifer Tidwell's pattern language, the first for UI design, offers many interaction patterns for GUIs (http://time-tripper.com/uipatterns/).

Web site design

A new book co-authored by Douglas K. van Duyne, James A. Landay, and Jason I. Hong, *The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience* (Addison-Wesley, 2003), offers 90 patterns targeted at Web site design (www.designofsites.com). For example, Quickflow Checkout explains how to let customers easily purchase items in their Shopping Cart without being delayed by unnecessary distractions such as Cross-Selling and Up-Selling.

UBICOMP DESIGN PATTERNS

The next step in the evolution of design patterns is to apply them in a more formative field such as ubiquitous computing by documenting lessons already learned in this new field or that can be carried over from previous design knowledge. Some of the ubicomp patterns we have brainstormed include Context-Sensitive I/O, Physical-Virtual Associations, Global Data, Proxies for Devices, Follow-Me Display, Appropriate Levels of Attention, and Anticipation. Some of these patterns focus on user interface aspects, some on systems aspects, and some on both.

Context-Sensitive I/O

One design pattern we have developed that hints at what an eventual ubicomp pattern language might be like addresses the following problem:

> Ubicomp devices will be used in a variety of locations and situations, but the device interfaces must not interrupt or distract the user from performing a primary task or annoy a nearby group of people.

Because certain input and output modalities are more appropriate in different circumstances, applying the design-pattern concept to this problem might yield the following solution:

> Input and output modalities should adapt to the user's current context.

For example, relying on speech or audio output is not a good idea when the user is participating in a meeting, attending a lecture, or in a movie theater. As Figure 2 shows, a context-sensitive cell phone should know when its owner is in a meeting and switch automatically to a vibration alert rather than require the owner to turn off the audible ringer.

On the other hand, direct manipulation input to a handheld device may be inferior to speech or pushing a few physical buttons when the user is driving a car. Likewise, when the driver places or receives a call, the car stereo volume should lower automatically.

Physical-Virtual Associations

Another ubicomp design pattern we have developed addresses the following problem:

> When people get together to collaborate in some way, they should not have to spend lots of time configuring their devices.

For example, at a meeting, the appropriate files, including biographies and contact information, should appear automatically on each person's laptop or PDA. The following solution would apply to this problem:

> When users are near one another, make it easy for their devices to connect and create an association that lets them share information over the life of a session.

A Context-Sensitive I/O device provides the appropriate output for making a Physical-Virtual Association. Some associations can occur automatically when two known devices come into physical proximity—for example, a user's PDA and PC could synchronize automatically. As Figure 3 shows, creating other associations might require direct user action—for example, letting others in the same meeting see documents on a PDA might require the owner's approval.

In other circumstances, users can connect their devices to create an association that allows them to share information over the life of a session. For example, the Hummingbird establishes a virtual connection between nearby users skiing at the same resort, enabling them to track one another's location on the slopes and thereby communicate more easily.

Applying design patterns to ubiquitous computing raises several interesting research questions. For example, how do you validate such patterns, given the time and expense required to test each one? Also, how do you evaluate the process of using patterns? Conducting controlled studies is difficult because of the creativity and skill involved in the act of design.

The process of developing design patterns is still fairly ad hoc, but we follow one simple rule: Find patterns that have been used successfully in real products or systems. Ideally, we prefer to find three good examples of a pattern being used in practice before we declare it to be a design pattern. We invite readers to submit their own ideas at http://guir.berkeley.edu/wiki/ubicomp. ■

James A. Landay is an associate professor of computer science in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. He is also on the faculty of the university's Group for User Interface Research and cofounder of the Berkeley Institute of Design. Contact him at landay@cs.berkeley.edu.

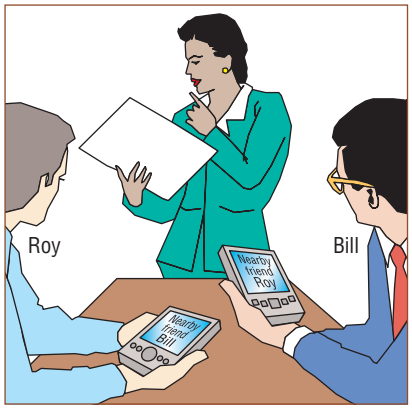

Figure 3. Physical-Virtual Associations design pattern. Devices should create associations, either automatically or via direct user input, when users are near each other.

Gaetano Borriello is a professor in the Department of Computer Science and Engineering at the University of Washington and laboratory director of Intel Research Seattle. Contact him at gaetano@cs.washington.edu.

Editor: Bill N. Schilit, Intel Research Seattle; bill.schilit@intel.com.