Design Challenges and Principles for Wizard of Oz Testing of Location-Enhanced Applications

Using Wizard of Oz techniques, designers can efficiently test location-enhanced application prototypes during the early stages of design, without building a full-fledged application or deploying a sensing infrastructure.

Yang Li
University of Washington

Jason I. Hong
Carnegie Mellon University

James A. Landay
*University of Washington
and Intel Research Seattle*

Location-enhanced applications adapt their behavior to or provide information about the location of people, places, and things. Boost Mobile's Loopt service (www.boostmobile.com/boostloopt), for example, lets mobile phone users identify their friends' current locations, while Navitime's EZNaviWalk (http://brew.qualcomm.com/brew_bnry/pdf/press_kit/navitime.pdf) helps users find optimal routes to their destinations based on the mode of transportation they select. Such location-enhanced applications are the most widely adopted type of ubicomp application; analysts predict that their use will grow tremendously in the near future.[1]

The problem, however, is that designing location-enhanced applications is often difficult.[2] Researchers have built toolkits for developing these applications,[3–5] but using the toolkits requires significant technical expertise. This makes it hard for interaction designers who lack a programming or hardware background to prototype, evaluate, and iterate on designs. In addition, location infrastructures aren't always available—GPS, for example, doesn't work indoors—and location-sensing technologies are still nonstandard and complex. In short, it takes significant effort to deploy a location-enhanced application so that you can realis-tically test it with users. And, by that time, it's often too late and too expensive to make major changes.

Here, we discuss Wizard of Oz techniques[6] for testing location-enhanced applications. WOz techniques are often employed in user interface (UI) prototyping and have been successfully used in several tools for early stage design.[7] The WOz approach lets users try out a system before it's fully developed. It accomplishes this using a "wizard" to simulate system parts that require sophisticated technologies, such as speech recognition[7] or location sensing.[8,9] This makes it easy to quickly explore many ideas because designers are less constrained by technical details.[10] Previous research has performed field experiments with location-based systems.[8–11] Here, we focus on tool support for WOz testing of location-enhanced applications, which pose two new challenges:

- They must incorporate location contexts, which have a much larger design space than traditional GUI input.
- The target setting is often a dynamically changing field environment, rather than a desk.

To address these issues, we built Topiary, which includes a suite of WOz techniques for testing location-enhanced applications. We previously described how designers can use Topiary to prototype and test location-enhanced applications.[9] Here, we use it as a proof of concept of WOz techniques and highlight three important research issues related

to designing WOz testing techniques for ubicomp applications:

- designing visual languages for WOz testing,
- allocating tasks between wizards and designers, and
- automating wizard tasks.

Topiary overview

Topiary provides a set of high-level prototyping abstractions—maps, scenarios, and storyboards—that designers can use in the tool's active map, storyboard, and test workspaces.

Designers use the active-map workspace to create a model of the location of people, places, and things. They can then place graphical objects representing these entities on a map to signify their locations and spatial relationships to one another—for example, "Alice is in the café" and "Bob is far from Tom."

In the storyboard workspace, designers can capture location contexts as scenarios. To create interface mockups, designers sketch pages that represent screens and create links that represent transitions between pages. They can then use captured scenarios as link conditions or triggers. For example, the designer might specify that an application prototype automatically switches from one page to another when "anyone moves near Bob." Likewise, when users click on a button, the prototype could show different information depending on the user's location.

Designers can use Topiary to test a design with real users by running the interface mockup on a mobile device, such as a PDA (see figure 1a). During a test, users interact with the interface mockup, while a wizard follows them and updates the location of people and things on a separate device (see figure 1b). The test workspace has two components:

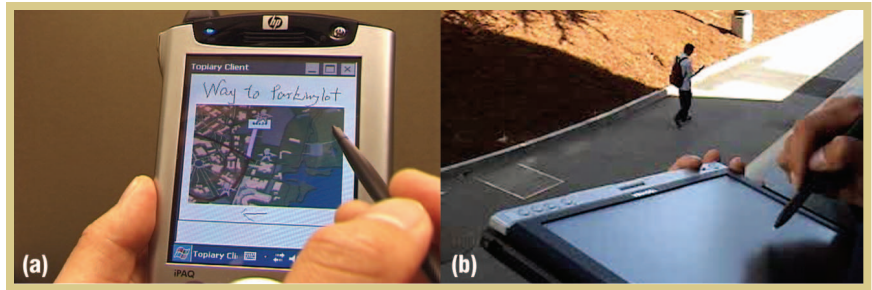- the *end-user UI* that users see and interact with (see figure 1a), and

Figure 1. A typical location-based WOz test setting. (a) An end user interacts with the end-user UI on a device, such as a PDA. (b) A wizard follows the user and updates his or her location in the wizard UI on a tablet PC.
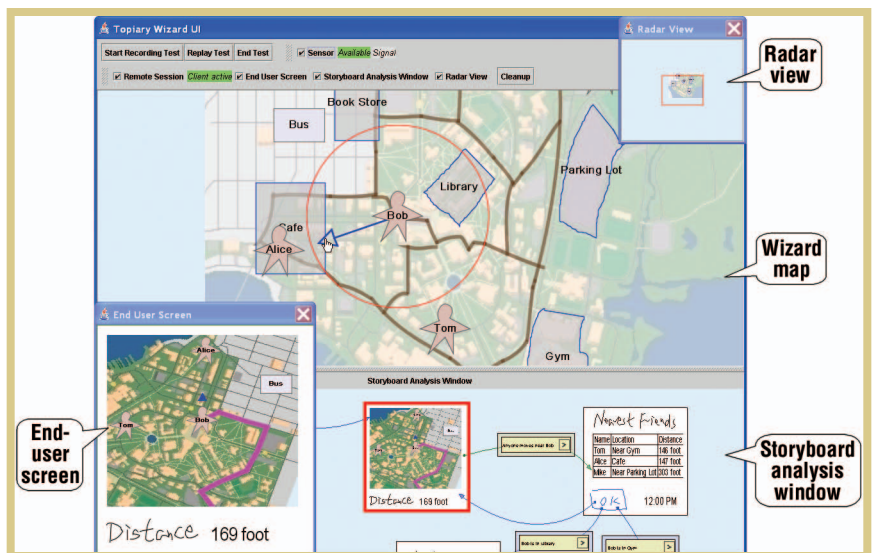

Figure 2. Topiary's wizard UI. The wizard map represents entities' current location and orientation; to simulate updates, a wizard drags them on the map. The end-user screen lets a wizard monitor the user's action on the end-user UI. The storyboard analysis window lets the wizard debug the interaction flow by highlighting the current page and the most recent transition. The radar view provides an overview of the wizard map. In this example, a wizard drags Bob's arrow to reflect his orientation; the map in the end-user UI (and on the end-user screen) automatically rotates so that its orientation is always consistent with the direction Bob is facing.

- the *wizard UI*, where designers simulate location contexts (see figure 2).

A wizard simulates location contexts by moving people and things around to dynamically update their locations. If moving a person or a thing activates a storyboard transition, the end-user UI automatically transitions to a target page. To simulate the entity's orientation change, the wizard rotates the arrow attached to the entity. A wizard navigates in the wizard map by either panning the map directly or circling a target region in the radar view.

Designing visual languages for WOz testing

Understanding how to design more effective WOz testing interfaces is one of our major research goals. In our view, to help wizards better run user tests and better simulate ubicomp applications, we need more appropriate visual languages. Traditionally, researchers address
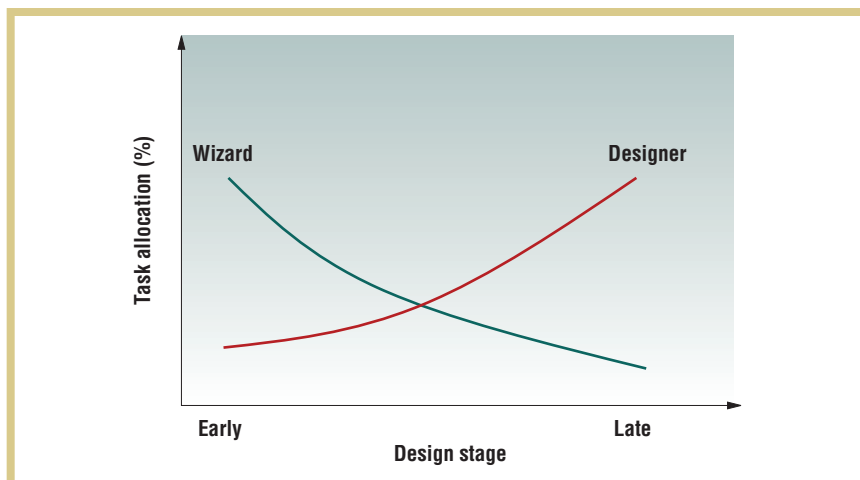
visual language issues only for creating a design. For ubicomp applications, however, designing visual languages for testing is just as important.

In the iterative design process, wizards and designers play two different roles, which either the same person or different people can perform. Ideally, WOz testing requires a wizard to update the prototype's state on the basis of the world state. A visual language for testing should help a wizard observe dynamic changes in a test environment and easily specify these changes to update the prototype's state. For example, in location-enhanced applications, dynamic changes refer to the movement of users and physical objects. In Topiary, the prototype's state includes

- the locations of entities stored in the prototype's repository (see the wizard map in figure 2), and
- the current storyboard page (see the storyboard analysis window in figure 2).

Most of the time, a wizard updates only the location of entities and Topiary automatically updates and maintains other parts of the prototype's state (for example, the current page) on the basis of the storyboard's interaction logic. To meet the goal of having wizards update prototypes on the basis of the real-world state, we propose two basic design principles. A visual language should

- make it easy for a wizard to efficiently

visualize the prototype's current state and perceive the difference between that state and the real-world state, and
- provide an interaction vocabulary that lets the wizard easily update the prototype state on the basis of his or her direct observations.

Topiary's wizard map, for example, provides a simple overview of entity locations maintained by a prototype, which lets the wizard easily see where updates are needed. Wizards need only capture two things about entities: their position and orientation. Both are based on the wizard's direct observation, and he or she can change them through direct manipulation.

When we evaluated Topiary with seven researchers and interface designers, they rated the tool's wizard interface highest among all features.[9] Participants particularly appreciated how straightforward the wizard interface is. We also conducted six field experiments with four people to test various designs of a tour-guide application. While acting as wizards, we found Topiary's interface was effective at capturing users' movement while walking. Our participants confirmed Topiary's WOz testing effectiveness: they didn't realize their locations were being updated by a wizard, rather than by real sensors.

Allocating tasks

In designing Topiary's WOz interfaces,

we found that there was often a trade-off in allocating tasks between designers and wizards. On one hand, designers can quickly create rough designs with few details, placing the burden on the wizard to simulate more sophisticated behaviors. On the other hand, designers can shoulder more of this burden by initially specifying more detail and functionality up front so that it can be automatically executed at test time, making the wizard's job much easier.

For example, to show users the shortest path in Topiary, a designer can—at design time—draw a road network on a map (see the bold brown lines on the wizard map in figure 2) and specify a starting point and destination (such as Bob's current location and the parking lot, respectively). Then, at test time, Topiary automatically constructs a shortest path on the basis of Bob's current location (see the bold pink line on figure 2's end-user screen). Alternatively, a designer can leave this feature unspecified and have the Wizard draw lines representing the shortest paths on the end user's screen during the tests. Clearly, the former requires more work from the designer, while the latter requires more from the wizard. The latter also lets a designer quickly try out rough ideas. As a design evolves, designers need to solidify their design ideas into a design that is validated by user tests (see figure 3).

Given this, a WOz tool should let designers choose appropriate task allocations for wizards and designers as a design evolves, to ensure a smooth iterative design process. For example, a tool can offer designers multiple options for designing the same feature. Such options might vary in terms of how much work they require from designers and wizards, and how large a design/testing scale they can handle. For example, while having a wizard manually draw paths requires

Figure 4. Specifying sensing errors. (a) The Specify Sensing Errors dialog box. Topiary uses the sensing accuracy (24 meters) as the standard deviation for generating Gaussian noise. (b) The end-user screen. The translucent circle represents the region of Bob's possible locations. The circle's size dynamically changes according to the distance between the wizard's selected location and the location generated by the sensing-error model.
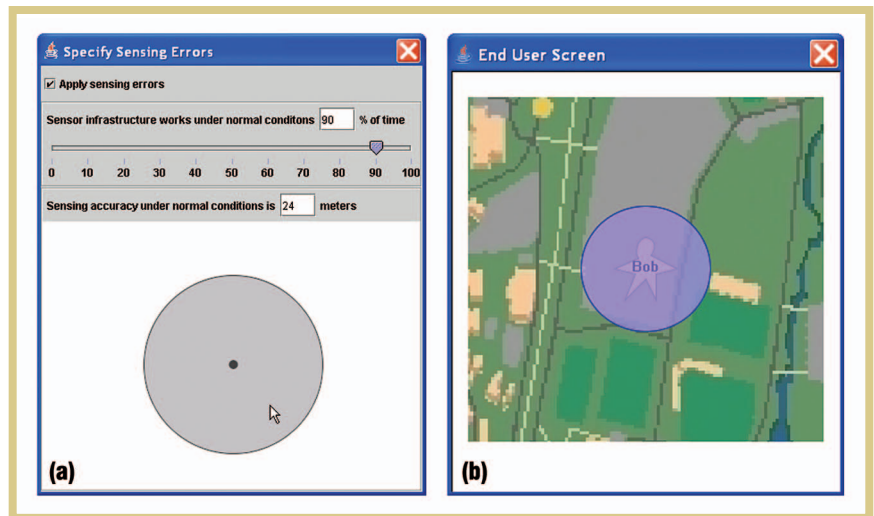
little work from a designer, it's very inefficient for a wizard to do in a complex testing environment. So, tools should let designers choose an appropriate approach based on whether they prefer quick iteration, larger-scale design, or something in between. This creates another design challenge for tool builders: how to present this redundant support to tool users so that they'll know which approach to choose for a particular design at a particular design stage.

Automating wizard tasks

It can be challenging for wizards to keep track of dynamically changing environments while testing ubicomp applications. We've therefore been designing and experimenting with various techniques to automate wizard tasks. Here, we discuss two such techniques. Although both are specific to location-enhanced applications, the techniques address two common problems of WOz testing of ubicomp applications: simulating sensing inaccuracy to better approximate realistic testing, and relieving wizards from routine tasks.

Simulating sensing inaccuracy

Sensing inaccuracy is an important issue in designing ubicomp applications. For location-enhanced applications, location acquisition depends heavily on sensed data (such as wireless signals, GPS, or accelerometer readings) and inferences (Place Lab uses temporal probabilistic reasoning, for example[5]). Sensed data is often noisy and sometimes even abnormal

owing to environmental variations or sensor hardware failure. Inferencing technologies are also imperfect. Consequently, location context is inherently ambiguous—a reported location isn't necessarily the user's real location, for example.

Simulating sensing inaccuracy can help designers uncover related usability issues in the early stages of design. In our previous Topiary version, a wizard could manually simulate sensing errors by dragging entities to a random map position. However, to lower wizards' cognitive load and systematically examine how sensing inaccuracy influences usability, we wanted to offer explicit support for modeling sensing inaccuracy.

Suede, a tool for designing speech-based UIs, lets designers specify speech-recognition accuracy and automatically generate random errors during a test.[7] Inspired by Suede, our new version of Topiary lets designers model location-sensing inaccuracy by specifying a target location-sensing infrastructure's stability and accuracy (see figure 4a). On the basis of such specifications, Topiary automatically generates location-sensing errors at test time using a simple probabilistic model. Generally speaking, it's hard to attribute a location-sensing error purely to sensor hardware or inference algorithms. So, we don't make distinctions in Topiary as to the ambiguity's source.

As figure 4a shows, a designer can specify sensing inaccuracy using the

Specify Sensing Errors dialog box. By checking the "Apply sensing errors" option, a designer can add noise to the wizard's simulation during a test. For example, when a wizard drags Bob to a map position, Topiary would automatically move Bob to another random position—such as five meters away—based on the generated noise.

Before testing, designers must specify how often a sensing infrastructure might work under normal conditions. This implies the sensing infrastructure's stability. In an abnormal condition, a sensing infrastructure gives completely unreasonable location reports. A designer can specify a percentage by either typing it in the text field or dragging the slider. During a test, the system uses this percentage to sample whether the simulated sensing infrastructure is in a normal condition. When it's not, the system randomly generates a map location based on a uniform probabilistic distribution, without considering the entity's location as set by the wizard. Otherwise, Topiary generates a random location around the wizard's location by applying Gaussian noise. Topiary automatically centers a translucent circle around a location generated by the sensing-error model on the end user's screen; this circle represents the entity's region of possible location (see figure 4b). The smaller the circle, the more accurate the location sensing.

Currently, we use a simple model for
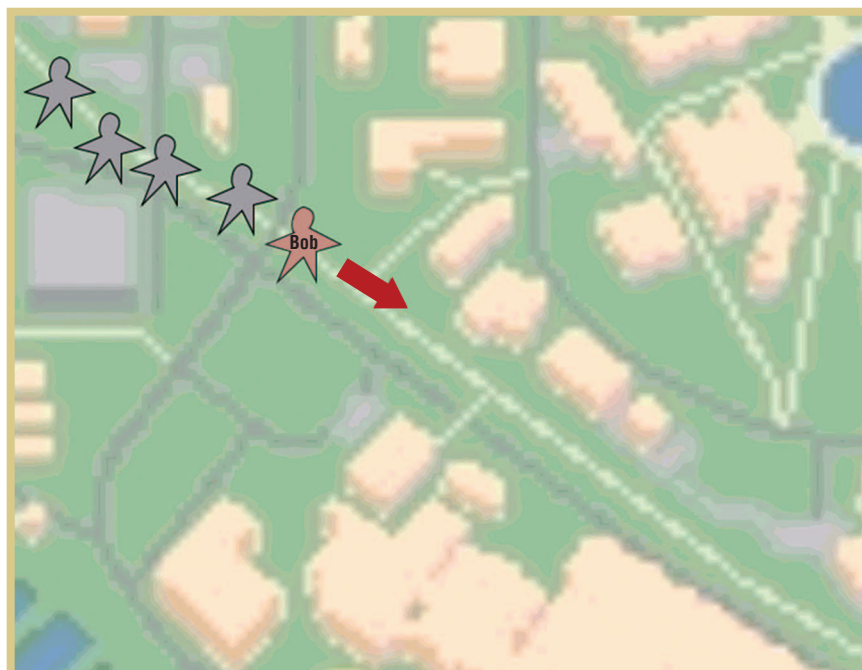
PERVASIVE computing

Figure 5. Topiary's cruise control. The tool can automatically update an entity's location on the basis of its current velocity. Wizards can deactivate the function by manually dragging the target entity. (The "gray star men," shown here to illustrate Bob's path, aren't visible in the actual interface.)

simulating sensing errors. Given the diversity of available sensing technologies as well as the physical world's complexity, our approach can't model all errors. However, we felt it was more important to make the model easy for nontechnologists to use and understand. Our approach also lets wizards simulate other location errors that might occur in a target situation by manually dragging an entity to a random position.

Wizards' cruise control

In our field study, we often observed that participants moved straight ahead at a constant speed, such as when they knew their destination and were walking down a street. To relieve wizards from routine updating tasks in such situations, we designed wizards' cruise control.

Topiary automatically infers an entity's velocity on the basis of its trajectory as produced by the wizard. Given this information, Topiary can automatically help a wizard update an entity's location. We designed this feature on the basis of the metaphor of automobile cruise control, which lets drivers maintain a constant speed without stepping on the accelerator or brake. In Topiary, a red arrow indicates an entity's velocity

during a test (see figure 5). The arrow's length indicates the entity's speed. A wizard might determine that an end user is moving straight ahead at a constant speed on the basis of observation of an entity's motion patterns and the testing environment's geographical attributes. In such cases, the wizard can turn on the entity's cruise control option, and Topiary will automatically move it at the current speed. If the wizard thinks the automatic update is inconsistent with the end user's current location, he or she can manually drag the entity to the correct map position. This mediation automatically turns off the automatic update, similar to how automobile drivers can deactivate cruise control by stepping on the accelerator or brake.

We're developing a new suite of tools for WOz testing. This work involves several ideas that researchers can extend to WOz testing of general ubicomp applications.

First, we'd like Topiary to support impromptu design during test sessions. Designers can use a WOz approach to rapidly explore various designs. In the

early stages of design, ideas are often vague—making it hard to cover all system aspects—while designers often have design inspirations while running a test. Given this, a WOz testing interface should let designers try out different output options and even create new feedback in response to unexpected discoveries made during a test. Building such a feature entails two challenges. First, how much flexibility should a wizard have? A WOz test shouldn't overload the wizard, and achieving flexibility should require little effort. Second, how can we capture and solidify impromptu designs so that they're not ephemeral?

Second, combining WOz testing with sensor-based testing will let users test their designs in more realistic situations. It will also enable longitudinal, large-scale design testing, which is useful for getting feedback and experience from users' daily lives rather than from controlled experiments. As a design matures, the need for wizards to mediate a test decreases, and sensor-based testing can play a larger role. Between the two extremes—complete WOz testing and complete sensor-based testing—the two approaches can work together. A tool should thus let designers transition smoothly from WOz testing to sensor-based testing as a design evolves.

Third, implementing multiwizard testing in Topiary would let users scale up testing when a target setting involves multiple entities and complex activities. For example, each wizard could use a separate device to observe and track a different entity's movement. However, the cost of multiwizard testing in a realistic situation is often high in the early stages of design. Consequently, it would

be useful—though challenging—to estimate a WOz study's complexity and automatically generate a plan for distributing tasks or roles across multiple wizards.

Finally, when designers evaluate a design, it's useful to analyze test data. So, a tool should be able to log both user behaviors and environmental changes. Topiary can capture user actions—such as mouse movements and clicks—as well as the physical paths traveled (whether simulated by wizards or generated by sensors). Because testing often generates large amounts of data, it's important to give designers an efficient UI to access this data. Currently, the only way designers can analyze tests in Topiary is to replay test logs. In the future, we'll support test data visualization by producing statistics of user–prototype interactions, such as how long a user took to find a correct location. We'll also enable more sophisticated visualizations for analyzing multidimensional test data. These additions will let designers efficiently analyze test data and collect feedback from a complex test setting. 𝓟

the AUTHORS
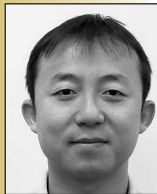
Yang Li is a research associate in the University of Washington's Computer Science and Engineering Department. His research interests include activity-based ubiquitous computing, rapid-prototyping tools, and pen-based user interfaces. He was a postdoctoral researcher in electrical engineering and computer science at the University of California, Berkeley. He received his PhD in computer science from the Chinese Academy of Sciences and is a member of the ACM and the ACM Sigchi. Contact him at the Univ. of Washington, Dept. of Computer Science & Eng., 506 Paul G. Allen Center, Box 352350, Seattle, WA 98195-2350; yangli@cs.washington.edu; www.cs.washington.edu/homes/yangli.

Jason I. Hong is an assistant professor in the Human-Computer Interaction Institute at Carnegie Mellon University's School of Computer Science. His research interests include ubiquitous computing, focusing on location-based services and usable privacy and security. He's an associate editor of *IEEE Pervasive Computing,* and he coauthored *The Design of Sites* (Addison Wesley, 2006), a book on the pattern-based approach to designing customer-centered Web sites. He received his PhD in computer science from the University of California, Berkeley and is a member of the ACM and ACM Sigchi. Contact him at the Human-Computer Interaction Inst., School of Computer Science, Carnegie Mellon Univ., 5000 Forbes Ave., Pittsburgh, PA 15213-3891; jasonh@cs.cmu.edu; www.cs.cmu.edu/~jasonh.

James A. Landay is a professor in the Computer Science & Engineering Department at the University of Washington, specializing in human-computer interaction. He was previously director of Intel Research Seattle, a lab exploring ubiquitous computing. His current interests include automated usability, demonstrational interfaces, ubicomp, design tools, and Web design. Landay received his PhD in computer science from Carnegie Mellon; his dissertation was the first to demonstrate sketching in UI design tools. Contact him at the Univ. of Washington, Dept. of Computer Science & Eng., 642 Paul G. Allen Center, Box 352350, Seattle, WA 98195-2350; landay@cs.washington.edu; www.cs.washington.edu/homes/landay.

REFERENCES

1. E.W. Pfeiffer, "WhereWare," *MIT Technology Rev.*, Sept. 2003, pp. 46–52.

2. S. Long et al., "Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study," *Proc. 2nd Ann. Int'l Conf. Mobile Computing and Networking* (MobiCom 96), ACM Press, 1996, pp. 97–107.

3. A.K. Dey, D. Salber, and G.D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," *Human-Computer Interaction*, vol. 16, nos. 2–3, 2001, pp. 97–166.

4. J.I. Hong and J.A. Landay, "An Architecture for Privacy-Sensitive Ubiquitous Computing," *Proc. 2nd Int'l Conf. Mobile Systems, Applications, and Services* (MobiSys 04), ACM Press, 2004, pp. 177–189.

5. A. LaMarca et al., "Place Lab: Device Positioning Using Radio Beacons in the Wild," *Proc. 3rd Int'l Conf. Pervasive Computing* (Pervasive 05), LNCS 3468, Springer, 2005, pp. 116–133.

6. N. Dahlbäck, A. Jönsson, and L. Ahrenberg, "Wizard of Oz Studies—Why and How," *Proc. 1st Int'l Conf. Intelligent User Interfaces*, ACM Press, 1993, pp. 193–200.

7. S.R. Klemmer et al., "Suede: A Wizard of Oz Prototyping Tool for Speech User Interfaces," *Proc. ACM Symp. User Interface Software and Technology* (UIST 2000), Computer-Human Interaction Letters, vol. 2, no. 2, 2000, pp. 1–10.

8. A. Krüger, I. Aslan, and H. Zimmer, "The Effects of Mobile Pedestrian Navigation Systems on the Concurrent Acquisition of Route and Survey Knowledge," *Proc. 6th Int'l Symp. Mobile Human-Computer Interaction* (MobileHCI 04), LNCS 3160, Springer, 2004, pp. 446–450.

9. Y. Li, J.I. Hong, and J.A. Landay, "Topiary: A Tool for Prototyping Location-Enhanced Applications," *Proc. ACM Symp. User Interface Software and Technology* (UIST 2004), Computer-Human Interaction Letters, vol. 6, no. 2, 2004, pp. 217–226.

10. S. Dow et al., "Wizard of Oz Support throughout an Iterative Design Process," *IEEE Pervasive Computing*, vol. 4, no. 4, 2005, pp. 18–26.

11. J. Goodman, S. Brewster, and P. Gray, "Using Field Experiments to Evaluate Mobile Guides," *Proc. Human-Computer Interaction in Mobile Guides*, Springer, 2004, pp. 38–48.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.