Position paper for the CHI 2001 workshop on "Building the Ubiquitous Computing User Experience"

Daniel Salber
IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532
USA
Email: salber@acm.org

Ubiquitous computing requires dramatic changes in systems development. Ubiquitous computing systems are distributed, use new devices for user interaction and rely on sensors to acquire environmental context information. Although distributed computing platforms have been a major topic of computer science research for a long time they need to be made relevant for ubiquitous computing. In particular, adequate software design abstractions, and design principles must be introduced and, as much as possible, factored into a supporting platform. If progress has been made lately to identify some relevant design abstractions, the issue of the design principles has not been addressed. Of particular interest is the elaboration of software design principles that embody good usability practices.

A similar process has taken place with graphical user interface software development. Software abstractions such as the event queue, or views and widgets have been introduced to manage the interaction with the user. These foundations capture some usability principles. Widget libraries (e.g., buttons, menus, etc.) manage user feedback. Many graphic engines are multi-threaded to allow for task interleaving and avoid system preemption over user actions. Grayed out text was introduced specifically to allow for error prevention. Menu and dialog boxes accelerators allow for flexibility, etc. Certainly usability design principles elaborated for graphical user interfaces are still relevant for interaction with ubiquitous computing devices and applications. However, additional design principles must be devised for ubiquitous computing, and adequate software support for these principles must be studied.

Indeed, ubiquitous computing raises new usability issues that warrant new design principles. In my opinion, three major areas are specific to ubiquitous computing, and must be explored: handling of failures, privacy, and sensor awareness.

Distributed systems researchers have developed strategies for preventing failures. However, most of these (e.g., redundancy) are unworkable or excessively costly for ubiquitous computing systems. Designers must factor in the possibilities of failures, and ubicomp infrastructures should provide adequate support. However, the question of how to handle failures from the user's point of view arises. The seamless access to distributed resources that the ubiquitous computing user experience aims for is jeopardized. How should failures be presented to the user? What repair strategies should be suggested to the

user, or implemented for automatic execution? What is graceful degradation in a ubicomp system and how should the user be made aware of it?

Privacy is a much discussed issue but few practical and efficient solutions have been proposed. Ubicomp systems should guarantee the confidentiality of and user control over personal data. How can a ubicomp system convince the user that it can be trusted? What are appropriate and workable privacy policies? How do we reconcile the need for data sharing between users, while enforcing sound privacy policies?

Finally, ubicomp environments often rely on a variety of sensors that sense the nature and availability of resources, but also the identities and whereabouts of people. How do we make the user aware of what data is sensed, collected for later use, and made available to other users and to whom?

Over the last few years, I have been developing software architecture models for managing sensed context information in ubicomp systems. I have also developed and deployed ubicomp applications using these models. The most significant effort I have been involved in, the Georgia Tech Context Toolkit, is a toolkit aimed at supporting rapid prototyping of ubicomp applications [1]. It handles component distribution and provides frameworks for acquiring and managing sensed context. It provides abstractions, most notably *context widgets*, to help with context-aware application development. I have worked on a number of applications built with the toolkit, using different sensors to determine e.g., the identity and location of users. I have recently extended the software architecture model of the Context Toolkit to account for meta-information associated with sensing (basically, features describing the quality of sensed context information) [2]. I believe I can contribute to the workshop in two ways: I have experience with ubicomp applications development and deployment in a community of users, and thus I have run into usability issues, some of which I have outlined in this position paper. Second, I am interested in investigating how usability principles can be adequately supported by a ubicomp infrastructure such as a toolkit.

[1] Daniel Salber, Anind K. Dey and Gregory D. Abowd. The Context Toolkit: Aiding the development of context-enabled applications. In Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99), Pittsburgh, PA, May 15-20, 1999. pp. 434-441.

[2] Phil D. Gray and Daniel Salber. Managing and using sensed context in interactive applications. To appear in Proceedings of Engineering for Human-Computer Interaction EHCI 2001, Toronto, May 2001.