From the Editor in Chief

Editor in Chief: M. Satyanarayanan ■ Carnegie Mellon and Intel Research ■ satya@cs.cmu.edu

The Many Faces of Adaptation

M. Satyanarayanan

Adaptive behavior is a recurring theme in pervasive computing. The need for adaptation arises when a significant mismatch exists between a resource's supply and demand. Often, the mismatch is in a low-level system resource such as network bandwidth, residual energy in a battery, CPU cycles, or memory. Sometimes it is a resource related to interaction such as display size or input modality. Or, in a context-aware system, it might be a high-level resource such as user attention or short-term memory.

ORIGIN

Why does the mismatch arise? Most frequently, it is due to mobility. For example, a user might move from one location where certain resources are plentiful to another location where they are scarce, or vice versa. Wireless network bandwidth is perhaps the resource most sensitive to physical location, since coverage is rarely uniform over a large area. The availability of nearby compute servers or data-staging servers is also location-dependent and therefore affects techniques such as *cyber foraging*.[1] In the case of code mobility, it is the application rather than the user that moves. In that case, both low-level resources and interactive resources such as display quality might vary widely between the source and destination systems. Resource variation can arise even when the user and code are static. Network bandwidth, for example, can vary widely depending on the actions of other users in the neighborhood. Sometimes just the passage of time causes a change in resource level. For example, energy becomes less plentiful for a laptop as its battery drains.

Regardless of cause, we can't simply ignore a gross mismatch between resource supply and demand. Doing so will result in an unsatisfactory user experience, usually because of sluggish system performance. Severe performance degradation can raise user frustration so much that it leads to increased human error, further increasing frustration. Sometimes, attributes other than system performance are affected. For example, a desktop application that has migrated to a handheld computer might present output that is unreadable on a small screen. As another example, ignoring the device's battery level could result in the user having to stop work prematurely. In all these examples, a system's proactive behavior—or "adaptation"—could improve the total user experience.

We can't simply ignore a gross mismatch between resource supply and demand. Doing so will result in an unsatisfactory user experience.

Adaptation is also important in the other direction, when resource levels improve. Otherwise, the user will be paying an opportunity cost by working in a "lean and mean" computing environment rather than a rich environment that is better able to bring his or her cognitive skills to bear on a particular task.

ADAPTATION STRATEGIES

Three strategies exist for adaptation in pervasive computing. First, a client can guide applications in changing their behavior so that they use less of a scarce resource. This change usually reduces the user-perceived quality, or *fidelity*, of an application. Armando Fox and his colleagues' work on image transcoding is one of the earliest examples of this technique.[2] Work by Brian Noble[3] and Jason Flinn[4] and their colleagues shows how we can broaden this approach across diverse applications and resources. We also can view changing the displayed output to match a small screen size as a form of fidelity change.

Second, a client can ask the environment to guarantee a certain level of a resource. This is the approach that reservation-based, quality-of-service (QoS) systems typically use.[5] From the client's viewpoint, this effectively increases a scarce resource's supply to meet the client's demand. To be viable in the real world, this approach must be combined with a framework of incentives for resource providers (such as a billing system) that encourages differential treatment of users.

Third, a client can suggest a *corrective action* to the user.[1] If the user acts on this suggestion, it is likely (but not certain) that the resource supply will become adequate to meet demand. We can view a corrective action as an executable *hint*,

Published by the IEEE CS and IEEE ComSoc ■ 1536-1268/04/$20.00 © 2004 IEEE

a concept now widely used in distributed systems.[6] Aura is an example of a system that is exploring this technique.[7]

All three strategies are important in pervasive computing. In a well-conditioned environment, a reservation-based approach might be feasible. At the same time, a user who travels extensively cannot assume that resource reservations will be supported everywhere. At some locations, reduced fidelity might be the only feasible form of adaptation. Using corrective actions broadens the range of possibilities for adaptation by involving the user. This approach could be particularly useful when lowered fidelity is unacceptable for a particular task.

IMPACT ON LAYERING

The need for adaptation complicates the decomposition of software into layers. Successful adaptation often requires merging information from diverse system layers. Unfortunately, layering can obstruct the visibility of critical information needed for adaptation. For example, suggesting a corrective action such as a short walk to a location with better wireless bandwidth is a high-level interaction with the user. However, wireless bandwidth is a low-level resource that is rarely visible to a system's upper layers.

Layering cleanly separates abstraction from implementation and is thus consistent with sound software engineering. Layering is also conducive to standardization since it encourages the creation of modular software components. Deciding how to decompose a complex system into layers or modules is nontrivial and remains very much an art rather than a science. The two most widely used guidelines for layering are David Parnas's principle of information hiding[8] and Jerry Saltzer and his colleagues' end-to-end principle.[9] However, these date back to the early 1970s and early 1980s, respectively, long before pervasive computing was conceived.

It therefore remains an open question whether layering and adaptation are truly compatible. We need sound design principles and well-validated implementation guidelines to help us structure software systems for pervasive computing. Until then, we will remain in the position of the ancient mariners who ventured into the vast unknown with maps of dubious accuracy.

To help us in this exploration, the current issue focuses on "Building and Evaluating Ubiquitous System Software." The guest editors, Vinny Cahill, Armando Fox, Tim Kindberg, and Brian Noble, all have extensive experience in designing and implementing software systems for pervasive computing. Their Guest Editors' Introduction describes the articles selected for this issue. P

REFERENCES

1. M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Personal Comm.*, vol. 8, no. 4, 2001, pp. 10–17.
2. A. Fox et al., "Adapting to Network and Client Variability via On-Demand Dynamic Distillation," *Proc. 7th Int'l ACM Conf. Architectural Support for Programming Languages and Operating Systems*, ACM Press, 1996, pp. 160–170.
3. B.D. Noble et al., "Agile Application-Aware Adaptation for Mobility," *Proc. 16th ACM Symp. Operating Systems Principles*, ACM Press, 1997, pp. 276–287.
4. J. Flinn and M. Satyanarayanan, "Energy-aware Adaptation for Mobile Applications," *Proc. 17th ACM Symp. Operating and Principles*, ACM Press, 1999, pp. 48–63.
5. K. Nahrsted, H. Chu, and S. Narayan, "QoS-aware Resource Management for Distributed Applications," *J. High-Speed Networks*, vol. 8, nos. 3–4, 1998, pp. 227–255.
6. D.B. Terry, "Caching Hints in Distributed Systems," *IEEE Trans. Software Eng.*, vol. 13, no. 1, 1987, pp. 48–54.
7. D. Garlan et al., "Project Aura: Toward Distraction-Free Pervasive Computing," *IEEE Pervasive Computing*, vol. 1, no. 2, 2002, pp. 22–31.
8. D.L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," *Comm. ACM*, vol. 15, no. 12, 1972, pp. 1053–1058.
9. J.H. Saltzer, D.P. Reed, and D.D. Clark, "End-to-End Arguments in System Design," *ACM Trans. Computer Systems*, vol. 2, no. 4, 1984, pp. 277–288.

IEEE Computer Society Publications Office

10662 Los Vaqueros Circle, PO Box 3014
Los Alamitos, CA 90720-1314

STAFF

Lead Editor
Shani Murray
smurray@computer.org

Group Managing Editor
Crystal R. Shif
cshif@computer.org

Senior Editors
Dale Strok and Dennis Taylor

Staff Editor
Rita Scanlan

Assistant Editor
Rebecca Deuel

Editorial Assistant
Brooke Miner

Magazine Assistant
Hilda Hosillos
pervasive@computer.org

Contributing Editors
Kirk Kroeker and Joan Taylor

Design Director
Toni Van Buskirk

Layout & Technical Illustrations
Carmen Flores-Garvey and Alex Torres

Production Editor
Monette Velasco

Publisher
Angela Burgess

Assistant Publisher
Dick Price

Membership & Circulation Marketing Manager
Georgann Carter

Business Development Manager
Sandra Brown

Advertising Coordinator
Marian Anderson

Submissions: Access the IEEE Computer Society's Web-based system, Manuscript Central, at http://cs-ieee.manuscriptcentral.com/index.html. Be sure to select the right manuscript type when submitting. Articles must be original and should be approximately 5,000 words long, preferably not exceeding 10 references. Visit www.computer.org/pervasive for editorial guidelines.

Editorial: Unless otherwise stated, bylined articles as well as products and services reflect the author's or firm's opinion; inclusion does not necessarily constitute endorsement by the IEEE Computer Society or the IEEE.

IEEE pervasive COMPUTING
MOBILE AND UBIQUITOUS SYSTEMS