



From the Editor in Chief...

The Interface Space

Robert E. Filman • RIACS/NASA Ames Research Center • filman@computer.org

“To use the computer as a transformation device is to use it on a trivial level. It is a completely general symbol-manipulating device and allows the writer of a program essentially to define what the machine is any way he or she chooses. That generality gives the computer a very special significance as the first modern device which allows itself to be used as a sort of do-it-yourself design kit, rather than as a single fixed-function tool.”

—Harold Cohen¹

Last issue, I complained about poorly designed interfaces and discussed the goals of good design. In this issue, I’ll try to bring a bit of regularity to the design space by cataloging the problem elements to consider in system-interface design.

Interface Space Dimensions

The design of a human-machine interface is (or ought to be) shaped by three classes of factors:

- the *task* to be performed and the facilities the system brings to performing it (that is, what you’re trying to do and what the underlying system actually does),
- the intended *users’* intellectual and physical characteristics, and
- the *channels* available for communicating with the system.

For any given design problem, some of these are fixed, whereas others can be varied by the system designer. In general, the designer’s aim should be to create an interface that facilitates the users’ communication with the system, over the available channels, to bring the system’s facilities to bear in performing the task.

Tasks

I remember when my eldest was little and loved to help. Of course, what daddy does most is type. Three-year-old Lisa would crawl into my lap and offer her services. When I suggested that she write

the next paragraph, she wanted to know which buttons to push. Of course, that wasn’t very helpful. Pushing the buttons is the easy part. Deciding which buttons to push is the part I really needed help with. While little Lisa was fun, she wasn’t really helpful. The moral of this story is that to be useful, systems need to help with some hard part of the task.

Usually, a system designer has some idea of the users’ goals — systems don’t evolve in pure intellectual vacuums. Unfortunately, many designers of computer systems confuse the components of a solution with the actual solution. That is, because the components provided can be assembled into a solution for the user’s problem, the designer thinks the problem is solved — a behavior we should perhaps call “Church’s interface fallacy.” Just because the given facilities can be combined to reach the user’s goal doesn’t mean that an interface is adequate. Users have primary sets of tasks or goals (“communicating messages among the members of my group,” “predicting the sales of our products under different pricing assumptions,” “writing the Great American Novel,” or “getting my clothes clean”). Providing only the components of the solutions does them a disservice.

In reality, the set of tasks we might want an artifact to aid in is unbounded, so we have a hard time characterizing them. Nevertheless, it’s useful to consider two task-oriented measures of system usability: depth and breadth.

A system’s *depth* measures how much of its anticipated tasks it accomplishes. Thus, a spreadsheet program used for predicting sales has a moderate depth — it has representations for sales and price values, but only primitive operators for using them for predictions. The spreadsheet is deeper than a pocket calculator, but not as deep as a system developed specifically for sales forecasts under pricing assumptions. A washing machine does a good job with part of the clothes-cleaning process, but it is too shallow to handle stains, dry cleaning, drying, or folding.

A system’s *breadth* is the extent to which it can

be used for lots of different things (including those that the designers failed to anticipate). Thus, the originators of spreadsheet programs intended them to supply a depth that mimics the calculations of accountants' spreadsheets. It is a tribute to the breadth of spreadsheets (and computers and programming languages) that they can be used in so many unanticipated ways.

For examples of uses unanticipated by system designers, consider that spreadsheets can be used to play Conway's game of life,² washing machines to churn butter if the farm is snowed in, and dishwashers to poach fish. (I, for one, have also had occasion to use a computer to heat my office.)

Decision Graphs

Given the variety of applications of artifacts, how can we characterize their tasks in some way that is useful for interface design? How does a system designed to help me write this column differ from one designed to help me

write the Great American Novel (and differ from one designed to help me write the Great Russian Novel)? (At a minimum, the character set, correct-spelling list, and hyphenation algorithms differ between Russian and English. Moreover, the Great American Novel editor resembles a familiar text processor, but when initially invoked, it inserts into your buffer, "It was a dark and stormy night.")

One useful tactic is to examine the tool's anticipated "use cases," considering what kinds of interactions should profitably exist, the various choices the user might want to make at each point, the relative likelihood of these choices, which parameterizations are required and where, and the extent to which we can anticipate the choices and parameters involved. We can build a graph representing these decisions, and its structure can lead us to interface-design tactics. Lean, low-branching-factor graphs suggest well-structured tasks through which a user can be led.

Bushy nodes imply many possible alternatives, and the best we can hope for is to make it easy to make choices. The former might be more characteristic of scriptable tasks, such as answering voicemail or setting up a VCR to record a future TV program; the latter, of less structured tasks like writing the Great American Novel. Things to keep in mind in this analysis include:

- **Predictability.** Although users might have the right to do lots of different things at any point, in reality, they tend to do some things a lot more than others. For example, in a messaging system, the user is much more likely to send or review a message immediately after composing it than to change his password. This is not to imply that the system should prevent users from changing their passwords at that point, but that it should make sending or reviewing the message easier to accomplish *at that point* than changing passwords.

IEEE INTERNET COMPUTING

IEEE Computer Society Publications Office
10662 Los Vaqueros Circle
Los Alamitos, CA 90720

EDITOR IN CHIEF

Robert E. Filman • filman@computer.org

ASSOCIATE EDITOR IN CHIEF

Li Gong • li.gong@sun.com

EDITORIAL BOARD

Helen Ashman • hla@cs.nott.ac.uk
Jean Bacon • jean.bacon@cl.cam.ac.uk
Elisa Bertino • bertino@cerias.purdue.edu
Scott Bradner • sob@harvard.edu
kc claffy • kc@caida.org
Siobhán Clarke • siobhan.clarke@cs.tcd.ie
Fred Douglass • f.douglass@computer.org
Stuart I. Feldman • sif@us.ibm.com
Ian Foster • foster@cs.uchicago.edu
Monika Henzinger • monika@google.com
Samuel Madden • madden@csail.mit.edu
Michael N. Huhns • huhns@sc.edu
Leonard Kleinrock • lk@cs.ucla.edu
Doug Lea • dl@cs.oswego.edu
Frank Maurer • maurer@cpsc.ualgary.ca
Daniel A. Menascé • menasce@verizon.net

Chris Metz • chmetz@cisco.com

Charles J. Petrie • petrie@nrc.stanford.edu
(EIC emeritus)

Krithi Ramamritham • krithi@cse.iitb.ac.in

Michael I. Schwartzbach • mis@brics.dk

Munindar P. Singh • singh@ncsu.edu

(EIC emeritus)

Craig Thompson • cwt@uark.edu

Steve Vinoski • vinoski@ieee.org

Dan S. Wallach • dwallach@cs.rice.edu

Jim Whitehead • ejw@soe.uscs.edu

IEEE Communications Society Liaison

G.S. Kuo • gskuo@ieee.nccu.edu.tw

STAFF

Lead Editor: Steve Woods,
swoods@computer.org

Group Managing Editor: Gene Smarte

Staff Editors: Kathy Clark-Fisher,
Rebecca Deuel, and Jenny Ferrero

Production Editor: Monette Velasco

Magazine Assistant: Hazel Kosky
internet@computer.org

Graphic Artist: Alex Torres

Contributing Editors: Cheryl Baltes, Greg Goth,

Keri Schreiner, Alison Skratt, and Joan Taylor

Publisher: Angela Burgess

Assistant Publisher: Dick Price

Membership/Circulation Marketing

Manager: Georgann Carter

Business Development Manager:

Sandy Brown

Advertising Supervisor: Marian Anderson

CS Magazine Operations Committee

Bill Schilit (chair), Jean Bacon, Pradip Bose,
Doris L. Carver, George Cybenko, John C.
Dill, Frank E. Ferrante, Robert E. Filman,
Forouzan Golshani, David Alan Grier,
Rajesh Gupta, Warren Harrison,
Mahadev Satyanarayanan,
Nigel Shadbolt, Francis Sullivan

CS Publications Board

Michael R. Williams (chair), Michael Blaha,
Mark Christensen, Sorel Reisman,
Jon Rokne, Bill Schilit, Linda Shafer,
Steven L. Tanimoto, Anand Tripathi



If one action is sufficiently likely, the designer can make it the default action, to be invoked by the simplest command (or even without explicit command).

- *Type constraints.* If we know what kind of parameters to expect, we can ease the task of specifying them. Default values, spelling correction, and completion are all interface mechanisms to aid in this activity.
- *Clustering.* While we might not generally be able to predict which command a user will invoke, certain applications find particular clusters or paths of commands.
- *Concurrence with other tasks.* If my routine consists of concurrently text processing and communicating (on the same equipment), then it is important that switching between these tasks be easy and inexpensive.
- *Functionality.* So far, we've blithely assumed that the underlying system is capable of doing whatever the task calls for. Having the desired functionality correctly implemented is crucial as, without it, all (or almost all) is lost.

Just as modeling some devices suggests more complexity than simple state automata, modeling devices with "universal actions" and concurrent subsystems might require more complex graphing techniques.

Channels

The various interfaces developed for a system must be strongly influenced by the quality of the system's input and output channels. The available channels are usually a given of system design, although the system designer can occasionally influence the hardware parameters. ("I need two LCDs to properly indicate the system state.") Channels vary in the following respects:

- *Permanence.* Most character and graphic output devices exhibit an intermediate degree of permanence — that is, what is written remains

on the screen until something new comes along to overwrite it. This contrasts with sound-based output, which the user needs to catch as it happens (or it's no longer there) and hardcopy output, which lasts indefinitely (or least until it's buried under something else on your desk). Another variety of indefinite information is embodied in the device's physicality. For instance, my computer doesn't need to periodically remind me which key is the shift key because it's labeled "shift."

- *Capacity.* Paralleling the permanence of output is capacity — how much information (in what varieties) can be shown at a given time. Capacity can be a mixed blessing, because a large capacity invites the interface designer to dilute a given communication's focus (recall our grad student from last issue who was intent on filling the screen with information).
- *Bandwidth and latency.* Latency refers to how long it takes before a system can respond. Bandwidth refers to how much information can be transferred in a given time period. Computers have gotten fast, limited only by interface designers' ability to invent things that take too long to do.
- *Precision.* On output, how precise a depiction can I create? Am I limited to letters at prespecified locations, or do I have high-density, color, bitmap graphics? Is the sound quality limited to single frequencies or symphonic? On input, can I read mouse coordinates, or am I restricted to recognizing keystrokes?
- *Accuracy.* How noisy are the input and output channels? Sufficient noise forces the interface designer to validate inputs and present output redundantly.

For applications running on general-purpose computers with bit-mapped screens, we've become accustomed to window, icon, menu, and pointing

device (WIMP) interfaces. It's either a tribute to the versatility of this scheme or a reprobation on the cleverness of interface designers that nothing has come along in the last 30 years to supplant this model. But interface design is about more than picking widgets for a WIMP — the most interesting new devices and applications need more mobility than that.

Users

Of course, the interface dimensions with the greatest variety are those concerned with people. People differ in cognitive, ergonomic, and social respects.

Cognitive dimensions include *memory*, the ability to remember the available choices and the information required to invoke different contexts. (In general, recognition is easier than recall, as long as you've got the channel bandwidth, capacity, and permanence to provide the prompts quickly enough and there's not so much to be recognized that it obscures or overwhelms.); *cognitive orientation*, such as a preference for pictures over words or structure over example; and *intelligence*, the ability to problem solve.

Ergonomics respects people's physical characteristics, such as the cost of precise motions or straining actions. Ergonomics teaches us that it's easier to move the mouse over a large button than a smaller one and that picking a menu at the top of the screen is easier than picking one at the top of a window (because you can safely overshoot the top of the screen but have to be more precise about the top of a window.) Careful ergonomic study can reveal that most people do not have three hands and that the transition from keyboard to pointing device is nontrivial.

Social dimensions include the environment in which an interface is used. Issues include *task expertise* (how skilled users are with the underlying task), expected *system expertise* (how skilled we expect users to become with the system — keeping in mind that everyone is a novice at some point and

that expertise is gained piecemeal, but that a system that always treats everyone as a novice grows tiresome), expected *use frequency* (how much users are likely to forget of how to use a device or feature between uses), and *social setting* (how much interpersonal help is available to aid in system use).

Closing Remarks

Orthogonal to our list of things to care about in interface construction are the components of interface devices. These range from command lines for typing in characters to the various widgets and components of the WIMP interface to interfaces based on exotic hardware. Similarly, interfaces can be organized as interrogations, direct manipulations of the some representa-

tional model, or blank canvases for the user to fill.

The best interfaces are the result of tinkering, testing, and evolution. It is unlikely that we completely understand our problem ahead of time; likewise, it's impossible to guess all the subtle interactions and multitude of ways that people can find to misinterpret a clever design. Interface prototypes must be built, examined, and tested, both informally and formally. Good interface development is expensive.

Interface developers have good reason to be overwhelmed with the number of possibilities. I've listed some of the interface-design space's many dimensions. Ideally, I'd love to present an organization of the universe that says, "If you have this kind of device,

used by the following kinds of users who do the following kinds of tasks, then the appropriate interface is..." but, I'm not that wise. All I can do in this column is suggest some of the things to look at and points to consider, so that you don't end up with a screen full of data when the user wanted a number, or a long dialogue leading to a predictable failure. Perhaps wiser souls will someday be able to organize these dimensions into a clear design space. □

References

1. H. Cohen, quoted in D. Michie and R. Johnston, *The Knowledge Machine: Artificial Intelligence and the Future of Man*, William Morrow and Co., 1985, p. 185.
2. B. Hayes, "Computer Recreations," *Scientific Am.*, vol. 249, no. 4, 1983, pp. 22-36.

IC Welcomes New Editorial Board Members



Helen Ashman heads the Web Technologies Research Group (WebTech) in the School of Computer Science at the University of Nottingham. Her research interests include hypermedia, particularly adaptive hypermedia, hypermedia modeling, and dynamic personalization in hypermedia delivery. She served as program chair of the 7th International World Wide Web Conference and the 2004 ACM Hypertext conference, served on the editorial boards of the *Journal of Digital Information* and the *New Review of Hypermedia and Multimedia*, and is on the ACM SIGWEB executive committee. Ashman received a PhD in computer science from the Royal Melbourne Institute of Technology (RMIT).



Kic Claffy is founder and director of the Cooperative Association for Internet Data Analysis (CAIDA), based at the University of California's San Diego Supercomputer Center, and adjunct associate professor in the Computer Science and Engineering Department at UCSD. Her research interests include measurement, analysis, and visualization of Internet workload, routing, topology, and performance data. CAIDA seeks, through the collection and curation of strategic Internet data sets and freely available tools and analysis methodologies, to improve the scientific integrity of network research and to promote more informed engineering, business, and policy decisions regarding the Internet infrastructure. Claffy received a PhD in computer science from UCSD.



Samuel Madden is an assistant professor in the Electrical Engineering and Computer Science Department at MIT and a member of MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL). His research interests span all areas of database systems; past projects include the TinyDB system for data collection from sensor networks and the Telegraph adaptive query processing engine. His current research focuses on modeling and statistical techniques for value prediction and outlier detection in sensor networks, as well as tools for managing intermittently connected networks of sensors. Madden received a PhD in computer science from the University of California, Berkeley.



Michael Schwartzbach is an associate professor at Basic Research in Computer Science (BRICS) at the University of Aarhus, Denmark. His research focuses on the design, implementation, and analysis of programming languages, XML, Web technology, and applications of nomadic second-order logic. He is chairman for the Danish External Examiners in Computer Science, and was an associate editor for *Theory and Practice of Object Systems*. He is an invited speaker for the 10th International Conference on Database Theory to be held in January 2005. Schwartzbach received a PhD in computer science from Cornell University.