The New Era of High-Functionality Interfaces

Henry Lieberman and Christopher Fry
Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA, USA

Elizabeth Rosenzweig
Bentley University
Waltham, MA, USA

Abstract

Traditional user interface design works best for applications that only have a relatively small number of operations for the user to choose from. These applications achieve usability by maintaining a simple correspondence between user goals and interface elements such as menu items or icons. But we are entering an era of high-functionality applications, where there may be hundreds or thousands of possible operations. In contexts like mobile phones, even if each individual application is simple, the combined functionality represented by the entire phone constitutes such a high-functionality command set. How are we going to manage the continued growth of high-functionality computing?

Artificial Intelligence promises some strategies for dealing with high-functionality situations. Interfaces can be oriented around the goals of the user rather than the features of the hardware or software. New interaction modalities like natural language, speech, gesture, vision, and multi-modal interfaces can extend the interface vocabulary. End-user programming can rethink the computer as a collection of capabilities to be composed on the fly rather than a set of predefined "applications". We also need new ways of teaching users about what kind of capabilities are available to them, and how to interact in high-functionality situations.

High-functionality (hi-fun) interfaces

As the range of tasks that people want to do with computers expands, and the capability of software grows, we are faced with the development of *high-functionality* (hi-fun) interfaces. We are not going to give a precise definition of hi-functionality interfaces here, but roughly, we mean those that provide large command sets, long menus, large or numerous icon bars, many data types, and complex patterns of use. In many cases, the names of interface operations may not be "obvious" to a beginning user, unless they know the underlying concepts of the application.

Low-functionality (lo-fun) interfaces are much simpler, acting on just a few kinds of data, and providing reasonably small command sets, where the name and effect of each command are expected to be immediately apparent to the user.

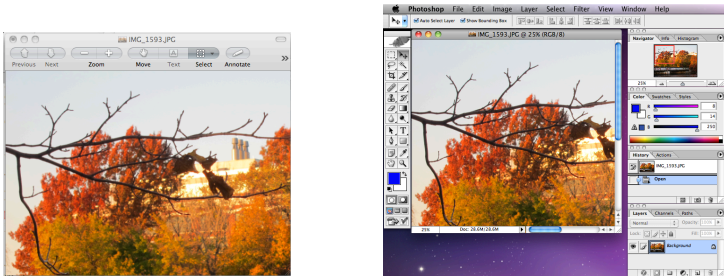

Figure 1. Apple's Preview (left) is an example of a "low functionality" application, with a few dozen operations. Adobe's "high-functionality" Photoshop (right) has thousands of operations.

Apple's *Preview* is an example of a relatively lo-fun application for images; it can, for example, print, crop, and rotate images, but it has relatively few operations (about 9 top-level operations, 7 menus of 5-15 items, few subsidiary dialogs). Adobe's *Photoshop* is a hi-fun image application (25 top level operations (+ modifier keys on many), 4 palettes of 2-3 tabs each, 8 menus of 10 to >25 items, many subsidiary dialogs). It has many different image types, and the total number of operations reaches into the thousands. It has a number of abstract concepts that it is necessary to learn, such as layers and different color models. It is user customizable, can record and play macros, has numerous plug-ins, etc.

Applications that become popular tend to grow into high-functionality interfaces over time as users desire more features and companies continually try to improve their products. The most successful, like Photoshop or Microsoft Excel, become languages

and programming environments in their own right. They become as powerful (and as difficult to learn for new users) as interactive development environments for programming languages.

The UI for high-functionality applications is typically designed for the expert and habitual user. It aims to make all the operations that the expert user would want to use easily accessible. But then the new user doesn't know where to start. And users who try to learn an interface by sequential exploration get confused because they are tempted to try many things for which they won't have use until much later, if at all.

User Interface Design for High Functionality Interfaces

Traditional user interface design aims to make interfaces easy to use, especially for beginning users, reduce error rates, and be aesthetically pleasing. The metaphor that most user-interface design tries to promote is that the computer is like a box of tools, like hammers and screwdrivers. Each tool is specialized for a certain job, and it's up to the user to know what each tool does, and which tool is appropriate for which job.

Interfaces strive for an ideal of simplicity and learnability. The menu items and icons that provide access to each tool should be suggestive of the functionality that the tool provides. Don Norman refers to this as "affordances" [Norman 99]. Simplicity demands a one-to-one correspondence between controls and tools.

The problem is, as the number of things the person wants to do with the computer grows, you wind up with too many tools in your toolbox. So you wind up with too many controls. Our screens fill up with icons and menu bars. Hierarchical menus, shift keys and other techniques, increase the capacity of the command set, but eventually space runs out (not to mention the user's patience), no matter what you do. The problem is that we're using the UI design principles that are appropriate for low-functionality interfaces to design the interfaces for high-functionality applications.

The alternative is what we call "goal-oriented interfaces". People have goals – things like "plan a trip" or "design a building". But programs don't have goals. What they have is specific functions, which are invoked by menu items, icons and typing. It's up to the user to figure out how to accomplish their goals in terms of the functions that the software provides. But when you have a large number of possible goals, and each goal might require a sequence of steps to be accomplished, planning the interaction becomes a high cognitive load on the user. This is at the root of most problems with usability of high-functionality applications.

The solution is to try to move as much as possible of the burden of translating goals into concrete functions, onto the computer. So Artificial Intelligence has a vital role to play. Natural language and speech recognition interfaces are a good way of

interacting with high-functionality applications, since a typical user can use the expressiveness of language to cover a wide range of possible goals. Gestural interfaces, image recognition, and other "natural" interface modalities can expand the vocabulary of interactive elements. Recent technical improvements in this area are making such interfaces increasingly practical. Traditional menu and icon-based interfaces can also be used for high functionality applications, but require a greater degree of context sensitivity and personalization to avoid overwhelming the user.

As an example of a high-functionality user interface architecture, Roadie [Lieberman and Espinosa 07] is a framework for goal oriented user interfaces. It provides a goal-oriented speech recognition interface to control a room full of consumer electronics devices like televisions, audio equipment, and other devices. The intent of Roadie is that the user can just speak their goal in relatively unrestricted natural language, and the system performs the operation.

The essential steps in Roadie's operation are:

- Goal recognition;
- Planning;
- Execution (either all at once or step-by-step); and
- (if necessary) Debugging.

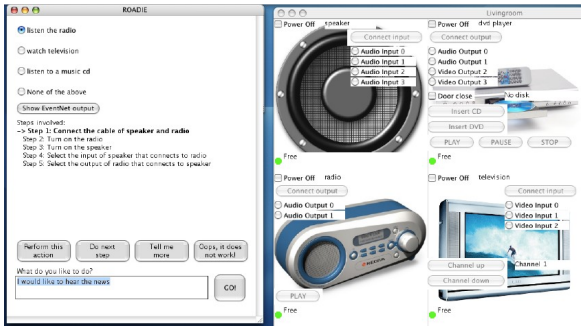

Figure 2. Roadie is a speech interface for consumer electronics. The user's utterance, "I want to hear the news" is recognized (top left) as one of a number of possible goals (TV, radio). For each goal, a step-by-step plan is constructed (middle left), and a variety of execution options provided (lower left). On the right, interfaces to each of the devices are simulated.

Roadie puts a variety of AI technologies to work, detailed in the reference [Lieberman and Espinosa 07]. Roadie uses a commonsense knowledge base to perform the goal recognition. It uses a partial-order planner to compute a plan from each goal. It has introspective knowledge of the capabilities of each device, mapping each function

onto possible goals. Diagnostic reasoning modules are invoked in case the user says "Oops" or is otherwise dissatisfied with the results. NLP, planning, and diagnostic reasoning are all well-studied AI technologies, and they result in enabling an easy-to-use, yet high-functionality interface.

The present and Future of Goal-Oriented Interfaces

Significant commercially available examples of goal-oriented interfaces are now beginning to appear. The most well-known of these is perhaps Apple's *Siri*, which provides a speech recognition interface to Apple's iPhone. In the same category is *Google Now*, which leverages the Google search engine to provide personal assistant functions. Microsoft also recently entered this arena with an agent named *Cortana*. IBM has ambitions to transform their successful Jeopardy-playing program, *Watson*, into a broad-spectrum user agent in a variety of vertical markets. These are undoubtably the first shots over the bow of an emerging category that will see rapid growth in the near future.

But at the present time, all these efforts lack some essential capabilities that would be desirable to make a personal assistant effective over a broad range of high functionality applications. The limitations of present efforts provide AI with a research agenda for improving the next generation of personal assistants.

Siri is only designed for one-shot speech interactions. You say something, and if Siri is able to recognize your goal, it invokes a single function in a single application. It can only work with a small number of applications programmed in advance. It can't engage you in a multistep dialogue, execute multistep procedures, or work with more than one application. It is purely a speech agent, and has no visual interface itself. Google Now leverages the high functionality Google search engine (and Siri has a tie to Wolfram Alpha), but again provides its personal assistant functions in a few well-defined areas only, indicated by its "cards", for things like reminders and route planning. There's no way to "debug" Siri or Now if they don't do what you want. Their capacity for true personalization is limited [Lieberman 14].

A key ingredient to making truly high-functionality interfaces is Commonsense knowledge reasoning. In interfaces, Commonsense knowledge can be used to provide intelligent, personalized, context-sensitive defaults; to adapt the interface to the user's expertise, goals, and particulars of a situation, and to provide proactive help. As mentioned above, Roadie consults a Commonsense knowledge base for goal recognition, planning, and debugging. The Open Mind Common Sense effort, which has been running for more than a decade, is collecting millions of statements of Commonsense facts [Lieberman et al 04], and we have developed a large number of applications in specific areas, from speech recognition to personalizing browsers and Web procedures [Faaborg and Lieberman 10], to enable high-functionality personal assistants.

No less a key ingredient is good user interface design. It is necessary to understand the needs of the user, understand how the user might use the interface to satisfy their goals, and get continual feedback from users on whether they are willing and able to learn, understand, and appreciate the functionality.

High-functionality Interaction is End-User Programming

Part of the key to removing these limitations is the realization that the user of a high-functionality interface is really engaged in what is essentially a programming task. After all, it's a computer; and what other way do we have a telling a computer what to do than to program it? We're just programming it with natural language (and perhaps pointing and typing and other modalities) rather than a conventional programming language. Programming is our high-functionality way of accessing the capabilities of a computer, and it's just that to date, we have artificially imposed on our users the low-functionality interaction paradigm of command and GUI interfaces.

So, we need some way to do everything that you might do in an interactive development environment (IDE). We need sequencing. We need functions with arguments. We need conditionals. We need loops. We need debugging. We even need capabilities that seem advanced or exotic in programming environments, such as introspection. AI also has a tradition of working on various kinds of "Automatic Programming" – generating programs from high-level specifications. Some work has also tried to bring together the commonalities between natural language dialog and programs [Liu and Lieberman 05]. Even though there is no conventional programming language, we're really doing end-user programming [Lieberman, Paterno, Wulf 06].

Justify [Fry and Lieberman 13] is an example. It is a high-functionality decision-support system for online deliberation and argumentation. Arguments are threaded discussions composed of "points", each of which has a type that indicates its role in the argument. It provides automatic summarization at every level, called "assessments", to emphasize their contingent nature. The type system is analogous to that of a programming language, and many programming facilities are provided. It even supports novel AI-style program generation, in providing a facility for Programming by Example [Lieberman 01].

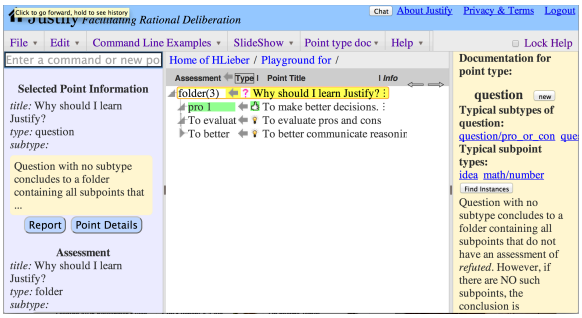

Figure 3. The *Justify* decision support system. It is a hi-fun system for online discussion, but really, it is a programming environment for decision support. It has a rich type system, and summarization and decision procedures can be programmed.

Justify has a total of 4808 interface operations, making it comparable to Photoshop (whose documentation index contains 4032 entries). While Photoshop has some programming facilities like macros, it doesn't take its nature is a programming system very seriously. Justify explicitly provides the programming operations and abstractions which give it the true generality that a high functionality system should have.

Learning High-Functionality Interfaces

Further, it's not enough just to give a user a high-functionality interface. They've got to be able to learn to use it. Today, many high functionality applications fail because beginning users have a hard time learning them. Some that are successful today are only that way because they started out as low functionality applications, and people gradually learned them over time. Photoshop 1.0 was actually a relatively low functionality application, roughly similar to Preview today. Today's gargantuan Photoshop is only acceptable because the community transitioned slowly to its increasing capabilities. If Photoshop had been first introduced in 2014, most likely it would have been rejected as too difficult to use!

The AI community has had a tradition of research into Intelligent Tutoring Systems (ITS), which hold the promise of being able to resolve the paradox of learning a high functionality system. Contemporary applications seem to have almost given up on the idea of help systems, after experience showing that conventional help is ineffective or that users ignore help. But in the new era of high functionality computing we need to revisit the idea of intelligent tutorials.

We have invented a new kind of interactive tutorial, the *steptorial* ("stepper tutorial") [Lieberman, Rosenzweig, Fry 14] that allows a learner to vary the autonomy of the interaction at every step. A steptorial is a kind of interactive tutorial based on the control structure of a reversible programming language stepper.

The idea is that the interface steps necessary to complete the introductory example are like a "program" (described by English sentences and/or interaction with the application rather than programming language code). The steptorial allows the user to step through the example, as a programmer steps through code. The steptorial is completely reversible. In extending the stepper metaphor beyond its origins in program debugging, we are enabling learning by end-user debugging of application use-cases.

Not having to choose a fixed level of autonomy in advance means that interactions can be tailored to the level of expertise of the individual user for that particular part of the application, supporting different cognitive styles. Depending on the situation at the moment, the user can choose help either in or out of context. Finally, having a variable level of autonomy reduces the risk, since the user can always go back and choose a different level of autonomy without any penalty.

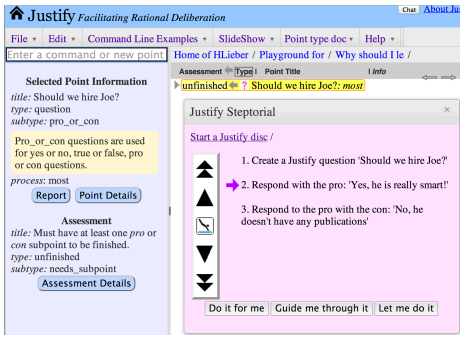

Figure 4. A *steptorial* ("stepper tutorial") for Justify. The user is stepping through a natural language description of an introductory example. At any point he/she can choose to have the system perform an operation, try it him/herself, be guided through it step-by-step [Kelleher & Pausch 05], or to back up to a previous state.

Again, we see this work in the AI tradition of Intelligent Tutoring Systems, and many opportunities arise for trying to better understand user behavior and provide personalized help.

Testing High-Functionality Interfaces

Finally, no discussion of user interface development is complete without attention to testing. User experience (UX) testers should realize that testing a high-functionality interface is not the same as testing a low-functionality interface. In low-functionality interfaces, UX testing may focus on the best way to organize the command set, receive user input, choose and display interface elements, and aesthetics.

In testing high-functionality interfaces, developers need more high-level feedback on the adequacy of the functionality, the flexibility and composability of operations, the learnability for new users, and the effectiveness of personalization to assure efficiency for expert and habitual users. Can users grasp the high functionality that the system can offer? This doesn't need to fully happen in the initial encounter; the new user need only get the sense that there is a world of utility waiting to be discovered. Can the users understand the essential concepts that will enable them to succeed on a simple but interesting example in the time plausible for an introductory session? Will they be enthusiastic about continuing to learn the interface as time goes on? How can the system communicate these essential concepts and offer assistance when users are confused or stuck? Does the system take into account different people's learning styles when providing help in learning the system?

Conclusion

High functionality interfaces are here to stay, and AI has a lot to contribute to the development of personalized interface agents that can operate over a broad spectrum of "application" capabilities that computers, phones, and function-specific devices can provide.

Today, some conservative or technophobic users recoil at the ever-growing proliferation of applications, worrying that the increased capability will be plagued by difficulty in learning and using interfaces. They blame themselves as being "non-technical" or too stupid to understand technology, when in fact, the fault is in poor interface design. Let's bust the myth that interfaces have to be "simple" and low-functionality to be easy to use. Human beings are enormously capable, and, at the same time, we have developed ways of interacting with each other that are effective and pleasurable. We should demand no less from our technology.

References

1. Faaborg, A., Lieberman, H., A Goal-Oriented Web Browser, in *No Code Required: End-User Programming for the Web*, Allen Cypher, Mira Dontcheva, Tessa Lau and Jeff Nichols, eds., Morgan Kaufmann, 2010.
2. Fry, C., Lieberman, H., Decision-Making Should Be More Like Programming, Int'l Symposium on End-User Development, Copenhagen, June 2013.
3. Kelleher, C. and Pausch, R., Stencil Based Tutorials: Design and Evaluation, CHI 2005,pp 541-550
4. Lieberman, H., ed, *Your Wish is My Command: Programming by Example*, Morgan Kaufmann, 2001.
5. Lieberman, H., Liu, H., Singh, P., Barry, B., Beating Common Sense into Interactive Applications, *AI Magazine*, Association for the Advancement of Artificial Intelligence, Winter 2004-2005.
6. Lieberman, H., Paterno, F., Wulf, V., eds. *End-User Development*, Springer, 2006.
7. Lieberman, H., Espinosa, J., A Goal-Oriented Interface To Consumer Electronics Using Planning And Commonsense Reasoning, Knowledge-Based Systems Journal, Vol 20, pp. 592-606, January 2007.
8. Lieberman, H., Say Hello to Smarter Apps That Fulfill Your Wishes, Wired UK, November 2012, p. 70.
9. Lieberman, H., Rosenzweig, E., Fry, C., Steptorials: Mixed-Initiative Learning of High Functionality Applications, ACM Conference on Intelligent User Interfaces (IUI-14), Haifa, Israel, February 2014.
10. Liu, H., Lieberman, H., Metafor: Visualizing Stories as Code, ACM Conference on Intelligent User Interfaces (IUI-2005), San Diego, January 2005.
11. Norman, Donald A. (1999). Affordance, Conventions and Design. *Interactions* 6(3):38-43, May 1999, ACM Press.