More than Skin Deep

Dan Dodge, CEO, QNX Software

Go back, way back, in computer history to 1990. You're sitting at a PC and you want to copy a file called letter.txt to a subdirectory called Letters. (Remember when folders were still called directories?) To perform this operation, you would type something like this:

copy letter.txt Letters

Pretty simple. In fact, some systems even provided auto-completion, so you didn't have to type the entire file name or directory name: Just begin typing the name, hit the Esc or Tab key, and the command shell would find the best match and fill in the remaining characters. Pretty cool.
Still, it didn't take long for the command line to vanish from the PC landscape. First Apple, then everyone else began to embrace the Windows-Icons-Menus-Pointer interface, aka the WIMP. The WIMP, we were assured, was more intuitive and more user friendly.

Or was it? Let me reconstruct my thought process the first time I used a WIMP to copy a file, some 18 years ago. The procedure, which still works with WIMP file managers like Windows Explorer, went something like this:

1. Double-click on the file folder icon to open the File Manager. <Wow, a cool-looking tool for organizing my files!>
2. Open the File Manager's Edit menu. <Huh? I wanted to copy the file, not edit it.>
3. From the Edit menu, select the Copy item. <Selected it... but nothing seems to happen.>
4. Double-click on the folder that you wish to copy the file to. <Okay, found it.>
5. Open the Edit menu again and click on the Paste item. <Oh... so that's how it works.>

Somehow, a simple operation grew from one step to multiple steps. Moreover, there was nothing intuitive about it. I mean, who would think of using an Edit menu to move or copy a file?

I am, of course, being unfair to the WIMP. After all, it does many things that the command line cannot. The point is, no single mode of computer-human interaction can address all user-interface challenges. Sometimes, the command line is best and sometimes the WIMP is. And for some systems, neither is best. Flatbed scanners, for example, were a mystery to many consumers until vendors wisely replaced some GUI-based controls with physical buttons like "Copy" and "Mail."

In fact, some systems do best with multiple forms of computer-human interaction, not just one. Consider, for example, an in-car infotainment unit that offers 3D navigation, realtime traffic reports, CD/DVD playback, and iPod connectivity. A voice-

controlled interface, with its ability to minimize driver distraction, is a natural choice here. That said, some functions will always be easier to control with a quick and simple button press. Thus, the system may also need a touchscreen, along with a few physical buttons.

But here's the thing. It isn't always easy to determine up front which functions should be controlled by voice, which by touchscreen, which by physical buttons, and which by some combination of the above. You must work closely with users to gauge which mode of interaction (or which combination of modes) works best and then fine-tune your interface accordingly.

This calls for a software architecture that not only supports multiple forms of user interaction, but also allows any feature to be controlled by a GUI one day, and by a voice interface the next. Simply put, you need an architecture that keeps your UI design options open.

You also need an architecture keeps that your UI available. The most brilliant user interface is useless if it locks up or becomes temporarily unavailable because the system is too busy doing something else. In a network, for example, a router that fails to provide performance data because it is swamped handling alarm conditions prevents operators from taking appropriate action. Likewise, if the HMI for a chocolate factory control system stops responding whenever the system experiences a high level of motor control, then operators can't take action if a critical event occurs. Fifty thousand ruined candy bars, anyone?

The more complex the system, the more likely such problems will happen. To avoid them, system designers must choose operating systems and middleware frameworks that can provide a guaranteed amount of CPU time and memory for user-interface functions, regardless of how busy the system becomes. Such resource guarantees can also thwart denial-of-service attacks and other network-based exploits that monopolize system resources and thereby prevent users from accessing the UI.

Put simply, to create a succesful interface for any complex product, you have to go beyond skin deep. You must be concerned not only with the layer that the user sees, hears, or touches, but also with the underlying software that ensures the interface is constantly available and quick to respond.
Because, after all, nobody likes wimpy response times.

Dan Dodge, CEO, QNX Software Systems