SEE TEXT ONLY

BY KEN MOORE

illuminating THE DESKTOP PARADIGM

What is a desktop? Is it just the end-goal of a graphical operating system (OS) such as Windows? Or is a desktop the graphical subsystems that can be paired with an operating system to provide application management and control? Is a desktop something that requires keyboard and mouse, or can a smartphone be considered a desktop? These are the kinds of questions that often arise when people discover that I am a desktop developer, but I think the essence of all these questions can be summarized in a single question: "Can a desktop be distinct from the underlying operating system?"

Apple and Android have taken the stance whereby the desktop is just one of several tightly integrated components that together provide a complete graphical interface for the end-user. This approach tends to reduce, or remove, any kind of text-based usage of the OS, greatly limiting the flexibility of that system. On the other end of the spectrum, we have the Unix-like operating systems, which typically have no built-in graphical capabilities and treat all graphical protocols as optional extras. So which approach is the best? (Figure 1)

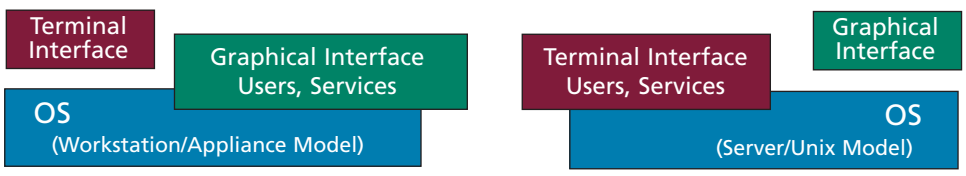
The answer to this OS-model conundrum probably lies in the purpose of the operating system and hardware combination for the end-user. From a distribution standpoint, a smartphone is treated as an appliance and typically allows a very limited set of functions. Because of this, it makes sense that the desktop be inseparable from the rest of the operating system since the desktop is not meant to be modified by the user. The same goes for Apple smartphones and laptops where the software is designed specifically for particular hardware, making the desktop nearly useless without the corresponding appliance. Where the boundaries start to become blurred is when you examine the more general-purpose operating systems, but overall there are two common approaches toward tackling the boundaries on these general-purpose systems.

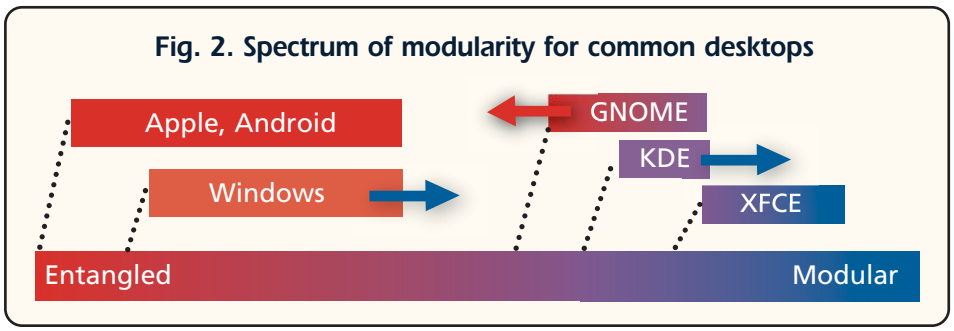
The Windows operating system exemplifies the first approach. It has historically been more like an appliance, where the desktop is almost completely entangled with the operating system. This approach has done very well as a graphical workstation OS for the past couple decades but tends to suffer in the headless-server markets due to the extra overhead for the graphical subsystems. Over time, Windows has been regularly working on untangling the interface from the OS and making the desktop components more modular so that they can enhance their functionality in the server market.

The second approach is exemplified by Unix-like systems. These have completely optional graphical components and correspondingly tend to do better as servers than as workstations. Almost all the desktops that are used with open-source operating systems today, such as KDE, GNOME, and XFCE, were born within this Unix-like environment. However, these desktops have typically striven to copy the Windows model and expect to be treated more like an appliance rather than a modular component of the OS. Figure 2 summarizes the major desktops and where I would place them on the spectrum of desktop modularity.

The design of the Lumina desktop emphasizes the benefits of a fully-modular desktop system and greatly differs from the other open-source desktops. We perceive that a fully modular desktop results in an operating system that can be used

Fig. 1. Common operating system models





for both server or workstation design goals with minimal disruption to the OS, as well as allow the desktop itself to be extremely portable between various operating systems. This is achieved by utilizing a standardized translation layer between the OS and the desktop but suffers from a higher level of code complexity in order to maintain this level of separation. To address this complexity, we have been developing an "OS Interface" class which acts as a completely modular, client-side API object that behaves as a dictionary for common high-level system interactions. By reviewing and categorizing all of the top-level OS interactions that a desktop may need, we are able to easily write and maintain a completely OS-agnostic translation layer that can optionally use OS-specific services and utilities without adding explicit requirements to the desktop itself. Table 1 lists the OS subsystems that Lumina interacts with for optional status indicators and such. Note how few are actually used in this type of desktop format, since the responsibility for OS-modification now falls to the OS, rather than the desktop, and can be as specific to the system as desired.

This type of arrangement results in a highly flexible desktop interface where all the OS interactions are both optional and readily configurable. All that is needed on the OS side to turn a traditional command line server into a graphical workstation is to install a few "desktop" packages such as standalone graphical tools for configuring the OS and the Lumina desktop for everything else. When paired with the flexible interface system that Lumina has also implemented (another subject for another article), a single operating system can power nearly all hardware configurations that utilize a graphical display.

The computing world is changing. The number of types of computer systems, both hardware and software, has been rapidly expanding, and operating systems must become more flexible in order to remain relevant. In my opinion, the BSD operating systems are particularly suited for this metamorphosis because they were designed with a minimalist but self-contained framework for the operating system and need relatively few adaptations to be converted between different types of end-user systems. The Lumina desktop is designed to work hand-in-hand with this type of operating system and together can expand into more graphically-focused market segments. With a continued focus on modularizing the underlying OS itself, brand new markets can be created that previously seemed impossible due to the shear amount of effort needed to create new "entangled" operating systems from the ground up. ●

KEN MOORE is the principal architect of the Lumina desktop and a software engineer at iXsystems Inc.

OS Subsystem	Read	Write	Optional External Tool
Batteries	Existence, status		
Audio	Current volume	Set volume	Full Audio Mixer
Network	Type from device name		Network Manager
External Media	Media shortcuts		
System Updates	Existence, status, logs	Start updates (on logout)	
System Power	Has permission	Start shutdown/reboot	
Screen Brightness	Existence, status	Set brightness	
CPU State	Supported, I/O		
Memory	Supported, status		
Hard Drive Status	Supported, I/O, % full		
Applications			App Store

Table 1: OS Interface interactions for the Lumina desktop