RTC
THE MAGAZINE OF RECORD FOR THE EMBEDDED COMPUTING INDUSTRY

BYOD – Bring Your Own Display

BY BILL WEINBERG, OLLIANCE CONSULTING – JANUARY 2013

Today, BYOD—Bring Your Own Device—allows or even encourages corporate users to leverage personal mobile phones and tablets to gain remote access to company assets and enterprise apps. BYOD is all the rage in both embedded and enterprise IT circles. Increasingly ubiquitous smartphones, tablets, Internet-enabled TVs and other "smart screens" are extending the BYO paradigm to domains beyond enterprise. In particular, if you find yourself designing almost any device with a user interface (UI), you can either embed an LCD screen directly in the device or let users BYOD—Bring Your Own Display.

This paradigm is not new. I advised developers/customers to embed HTTP servers in their products as far back as 1996, as a means to present a rich and reprogrammable UI in lieu of or together with dedicated display hardware. But 16 years ago, most devices deployed 16-bit CPUs running embedded OSs lacking TCP/IP or networking of any kind. By comparison, today's embedded hardware boasts 32- or 64-bit silicon, deploys Ethernet or Wi-Fi, and runs Linux, Android or a full-function RTOS. And unlike device operators of a decade ago, today's users don't need a desktop PC to support a web UI. Instead, they carry around richly provisioned display-capable web clients—mobile handsets and tablets.

Another big difference in this brave new world of BYOD is that originally, HTML and Java device UIs often represented "second class" user experiences (UX) compared to native GPU-enabled ones (excepting "lumps on a line" (LoaL) devices like routers and access points). Conversely, today's web-based UIs are where developers put their best feet forward, with device-local displays reduced to a bare minimum (e.g., the simple, elegant UI on the NEST thermostat itself vs. the fancy mobile NEST UI presented to iPhones and web browsers). This shift in emphasis is predicated by cost (browser pixels are cheaper than liquid crystals) and by the ability to present a superior UX using Flash, HTML5 and native apps purpose-built to run on remote mobile devices under Android or iOS (Figure 1).

Figure 1 – Evolution from fixed/integrated display to BYO.

Deciding When to BYO Display

When embarking upon new design projects, device developers should think seriously about where to make their UI investments. Start by asking whether the device requires device-local intervention. Do common usage scenarios involve users working in close proximity (a kitchen appliance, an automotive/IVI dashboard, a high-speed milling machine)? If not, then the design is a strong candidate for BYO Display. Even if a device does require an integrated console, it can still benefit from a secondary remote UI for diagnostics, pre-programming, remote control, etc. And if the local

intervention occurs only through a reset switch or a big red STOP button, all the more reason to remove the remaining UX to a remote device.

Another consideration is whether it will participate in a local area network (LAN).If the device is already provisioned with Ethernet or Wi-Fi, then it's a trivial step to embed a web server (Apache, TinyHTTP, etc.) to enable BYO Display. Or is it going to connect to a WAN or the Cloud? Many locally networked devices also benefit from connection to points beyond the firewall. Examples include telemedicine systems, premises monitoring and automation systems, storage appliances and others. The entire class of M2M clients falls into this category as well.

Deciding whether to follow the BYO Display route or embed a local display requires considering a number of functional and financial factors. These considerations were driven home to me over the last year during specification of an in-home display (IHD) system for residential and commercial energy monitoring and premises control, from which I will draw arguments and examples.

Interface Latency and Locality of UI

An important rule of good UX design is that users should receive immediate feedback and presumably near-immediate responsiveness from UI input. In the case of our IHD, a user might want to turn on or off a set of lights or engage an air conditioner or heat pump. The reference UX is the legacy light switch or thermostat. "Click"—users expect a light to illuminate instantaneously, and heating and cooling systems to respond audibly after a few seconds. Device-local UIs and LAN-based BYO Display can deliver a comparable UX even on slow hardware and low bandwidth control and communications networks.

An increasingly common use case is premises monitoring and control in the Cloud. If a user wants to analyze energy usage, turn lights off or turn air conditioning on remotely, then latency is not a huge issue—delays of seconds or even minutes induced by Cloud implementation will not greatly impact the UX. But if a user demands an immediate response from a Cloud-based control system, all bets are off. Fast response scenarios include real-time temperature monitoring, fire detection, and control on behalf of a third party, for example, turning up or down the heat in your mother's apartment while she is on the phone with you.

Cloud-based implementation can have strong promoters in the supply chain. For example, an energy supplier may offer big rate breaks to premises owners who opt in to a combination of Smart Metering and IHD-based monitoring and control. A utility's priority is clearly data gathering and billing, not user experience, predicating a Cloud-based implementation over a more user-friendly local one.

Mission-critical and life-critical applications tilt in the direction of device-local or LAN-based BYO Display. While it's tempting to try and control a chemical plant or check on patient vital signs from a smartphone while sitting on the beach, extremely bad outcomes can arise from user isolation and unbounded interface latency.

Security

A BYO Device in the enterprise is cause for concern and debate among security wonks and IT managers. Similarly, security is a key concern when thinking about when or if to implement remote BYO Display UIs. Both device-local and mobile client UIs present security risks.

Local displays engender primarily physical security concerns. If a system console is exposed to human access, it is subject to cracking and to physical attack. Systems with local displays also can have network interfaces and potentially exposed I/O ports, presenting entry points and attack surfaces.

On the other hand, remote UIs expand the security perimeter and/or the extent of vulnerability. The primary device can devolve to a more defensible and concealable "lump-on-a-line," but the networked UI is then exposed to exploits of data in transit, man-in-the-middle attacks, DoS attacks, spoofing the UI device ID, and cracking the web or native app UI on the remote/mobile device.

The BYO Display paradigm, if applied judiciously, can reduce the attack surfaces to 1) a single TCP/IP port and 2) the web interface itself. In today's hostile network security environment, no device should come to market that is not locked down and subject to security scrutiny as part of test and QA processes. Otherwise, the remote UI can become the point of entry for buffer overflow and other browser-based attacks.

Open or Closed Systems

Embedded systems, with and without displays and UIs, were originally more or less closed, mono-function devices. The application encompassed the entire device and device programming did not change over deployment lifetimes. With the advent of application-level OSs, Linux and Android in particular, all types of intelligent devices can become open systems, amenable to deployment of new applications with various options for changing firmware and system software as well.

The open vs. closed distinction impacts how and if designers implement BYO Display. If you want to build a system that is open to user-mediated deployment of application-level software, then you are likely to build your device on Android. Android and Linux both can easily host web servers and run back-end web apps. But Android applications from Google Play, the Amazon AppStore for Android and other channels, only run and output on a device-local UI (think phone and tablet-based LCD screens). You could also integrate a web-based BYO Device UI, but it would not benefit directly from the application platform nature of Android.

Conversely, for more static and closed designs, Linux or even a legacy RTOS would suffice, presenting a web-only UI and a UX colored by the richness of available browsers. Although such systems enjoy straightforward updates to the entire software stack, including the UI, there is no one single application development method nor a unified distribution channel for after-market apps. Application architectures and frameworks vary greatly across a rich if fragmented marketplace.

Cost / Device Provisioning

In our debate for the IHD design, this open/closed, local/remote divide led to two fairly distinct design paths, BoM costs and accompanying business models shown in Figure 2.

Figure 2 – Design path alternatives for local vs. remote user interfaces.

1. An Android-based device with an attached display, offering rich local functionality and running third party apps.
2. A Linux-only LoaL device with no display and a primary web-based UI.

Path 1 requires a more robust bill of materials (BoM)—in particular, the physical display itself, a graphics chipset or GPU, VRAM, local input devices (touchscreen, etc.), and to run Android effectively, a mid-level or better CPU.

In sharp contrast, Path 2 presents a much more modest BoM. It's a lump-on-a-line. There are no local display or input devices, no graphics chips or VRAM, and running just a Linux Apache MySQL Perl/PHP/Python (LAMP) stack or comparable requires less CPU horsepower and it demands less DRAM and flash storage—no apps to store and run concurrently.

Each has its virtues. Path 1 is more amenable to acting as a services host—the IHD can do more than just implement smart energy functions. It can act as a conduit for secondary capabilities (and revenue streams) like premises security and telemedicine and run third-party apps to enhance its core and secondary functions, but only via the fixed, local display.

Path 2 is much cheaper to build and deploy. However, given its more meager provisioning, it offers less local functionality and is best deployed as a conduit to/from Cloud-based infrastructure.

Client UI Architecture – Browser or Native App?

Until the introduction of the second-generation iPhone Appstore, and the launch of the Android Market (today called Google Play), remote UIs built on one of several architectures:

- HTML
- Java
- Flash
- X11 and Thin Client protocols (when available)

When iPhone apps replaced web apps as state-of-the-art, they captured the imagination of tens of thousands of developers and gave intelligent devices a new UI option—the mobile app. Today, device manufacturers increasingly ship their wares accompanied by iOS and Android apps as favored BYO UI implementations. At my house, my thermostat, my IP camera (to watch the puppies), my printer and my broadband router all boast smartphone and tablet apps. My neighbor's car, washing machine and sprinkler system do the same.

Mobile app development lies beyond the scope of this article, but the topic merits a few comments in terms of supporting BYO Display. For one thing, many iPhone and Android apps are just web apps encased in a native application wrapper. While native mobile apps today offer the greatest marketing cachet, they are more expensive to develop, test and maintain, especially across the fragmented Android portfolio of hundreds of devices and dozens of versions and forked

implementations. And now, with the growing popularity of HTML5, web apps are poised to make a comeback, especially for BYO Display, with the benefit of less fragmentation and greater interoperability across UI client device types, OSs and versions.

Web protocols, Cloud infrastructure and APIs, along with mobile apps, give device developers powerful tools for building UIs and crafting engaging user experiences. The BYO Display paradigm is a good option for almost all types of devices, but not necessarily as the primary UI. Careful consideration of design goals and use scenarios will reveal whether it makes the most sense to leverage readily available display surfaces on smartphones, tablets, TVs and desktop PCs, or whether a design calls for a local display and a heftier BoM.