Progressive Web Apps

By Bob Frankston

Increasingly, consumer electronics devices are software applications (apps). In the April 2014 issue of *IEEE Consumer Electronics Magazine*, I wrote about HTML5 as a programing environment (https://rmf.vc/iEEEHTML5). Today's progressive web applications (PWAs) go further. They take advantage of HTML5 and the JavaScript environment capabilities.

The term *progressive* refers to the approach of taking advantage of capabilities that are available in the application environment rather than having rigid requirements. This includes running a range of devices instead of focusing on narrow categories like the mobile-first approach. Moving across a disparate array of environments should include making use of opportunities that are available, such as a larger screen and adapting to the limits of a small screen. Perhaps in the future, we'll treat the screen as an optional interaction surface given the availability of voice, touch, and other methods.

Instead of grouping a set of disparate concepts under the term *mobile*, we can think of them separately:

- The screen is an interactive surface. It may be a small, wall-mounted device or a large screen on a device we carry around.
- Mobility is not simply a characteristic of a device; it should be thought of as the mobility of a person who can interact on any available surface rather than being limited to a device he or she is carrying. Applications can also be mobile and available where needed rather than on a particular device.
- We can introduce place as its own attribute. Touching a surface in the living room would turn on the light in that room. When we place a phone call, we may be trying to reach a person, or perhaps we are trying to contact anyone who happens to be home at the time.
- There are many interaction modes, with touch being just one. Cameras can enable rich gesturing or even the use of facial expressions. Voice is another interactive mode, and there are so many more.

While a PWA can be treated like a standard application on a device, the ability for it to be run from a uniform resource locator (URL) makes it easy to use the application on any device with a browser.

We've come a long way from the original iPhone with its then-rigid specifications. PWAs now give us a chance to explore new possibilities. While a PWA can be treated like a standard application on a device, the ability for it to be run from a uniform resource locator (URL) makes it easy to use the application on any device with a browser. This allows one to travel light. Today, airlines are removing screens from planes and expecting travelers to carry their own devices. Some airlines hand out tablets. Perhaps it will make sense to bring back the larger screens to connected devices with browsers.

APPS

Unlike desktop applications, which have had full access to the capabilities of the hardware (even more so in the earlier days of personal computers), a PWA starts out with essentially no access beyond the content from its original site. This reflects the cautious safety-first model for the web.

These capabilities are becoming more available to browser applications. They can now get the user's location (if permission is granted) and support Bluetooth devices (again, with permission). Soon they will be able to process payments. We can think more of PWAs as true applications and not just as cached web pages.

Google and others are using PWAs as an opportunity to enforce hypertext transport protocol secure (https) encryption. Central to the structure of a PWA is the *Service Worker*—typically a file in the top level (or other) directory of the application. A common name is */sw.js*. It is distinct from the rest of the application, and instead of a top-level window object, it has a *self*-objective. It is a kind of web worker (https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API) that communicates with the rest of the application by sending messages rather than directly referencing shared objects.

Digital Object Identifier 10.1109/MCE.2017.2776463
Date of publication: 8 February 2018

The service worker has limited access to files in or below its file directory. This is part of a larger approach of treating the URL path as a file system path, but sites can get creative with interpreting the string. This cautious approach also limits access to platform capabilities, such as the native file system.

Notification is a new key capability that relies on a service worker being available. Even when the browser is closed, the JavaScript engine is still running. The notification mechanism allows data to be updated in the background so you can quickly see the current state of orders or reservations even if they are not connected at the moment. The application can also use the platform notification mechanism to alert the user of changes. To prevent denial-of-service attacks, the notifications are routed through the browser-supplier servers.

There is also a local data store, IndexedDB, which advances over the browser local storage. In addition to having larger capacity and transactions, IndexedDB is asynchronous. Asynchronicity is a key feature of HTML5 programming, and the promise mechanism has improved the readability of asynchronous programs. This style can make applications very responsive while still having the simplicity of single threading.

I expect that more platform capabilities, i.e., calendar providers, will become available to PWAs. These concepts are being pioneered on mobile devices but also prove useful on desktop devices, allowing more integration with workflow.

LOOKING AHEAD

Because PWAs are constrained by the browser environment, it is hard to directly share among PWAs from different sources or across browsers. I would like to have the ability to share resources among browser apps and among local machines rather than relying on distant servers.

In theory, some capabilities can be provided by third parties with local https servers, and perhaps we'll see such offerings as document stores. There is still a need for a more traditional file system that only stores collections of bytes without knowing what they mean. This is similar to accessing a file system on today's secure digital cards allowing multiple programs to store image files so that multiple camera applications and photo-processing programs can share access to the same files and create new formats. How should access to such objects be managed? How do I selectively share access and limit others' ability to view a picture even when they have physical access to the device?

We need to shift to thinking of the Internet in terms of peer connectivity rather than as something that one can access.

Today, we have browser machines, such as Chromebooks, but their capabilities are limited by the available apps. The same hardware can be used in a Windows Netbook, which offers far more capabilities. With time, it may make more sense to have such browser-based machines without worrying about the underlying operating systems.

PWA ENGINES

PWAs offer the ability to write once and run everywhere. Smart TVs are an example of a market that is currently fragmented, as various vendors build their own smart TVs, along with boxes from Amazon, Roku, Apple, and others.

Having a browser box could provide a standard platform that brings the benefits of web technology to this market. Concepts like URLs would be used to bookmark content (formerly known as *TV shows*) so they can be shared. Such capabilities as windowing (picture in picture) can be implemented locally and flexibly.

Devices like home lighting and climate controllers can be implemented as applications using generic hardware and can then be available anywhere in the house. Voice services such as Amazon's Alexa and Google Home can take advantage of place and monitor conditions, such as temperature. (Alas, yes, they can also listen in, so proceed cautiously.)

PWAs AND INTERNET OF THINGS

The cost of a browser-capable computer keeps dropping. However, for many applications, there is need for running a browser engine on the device itself. Such devices still benefit from being expertise-developed for browser-based applications. NodeJS supports running JavaScript on a wide variety of machines. Onion.IO sells a complete system including Wi-Fi for US$7.50 retail (as of November 2017).

We need to think about an Internet of Things built of fully capable devices. There will be sensors and other devices that may be resource constrained, but those devices can be considered as peripherals to fully capable devices. Similarly, the programming environment of JavaScript is far more resilient than a low-level language like C.

Rather than a mobile-first approach, we can take a local-first approach to development. Thus, a doorbell (it's no longer a bell but a two-way communicating device) might take advantage of a face recognition service but can still function if the recognition service is not available. A PWA application should communicate directly with the door device rather than relying on a distant server to merely authorize the door to open.

The Internet protocols are not quite there yet. The domain name system depends on distant services, and the Internet protocol address is issued by a provider. We need to shift to thinking of the Internet in terms of peer connectivity rather than as something that one can access.

ABOUT JAVASCRIPT

JavaScript started as a low-performance scripting language that was designed in only one week. However, it was built upon a long heritage of important design principles where the focus was on safety and resilience. Objects were given dynamic definitions instead of having static class definitions.

(continued on page 117)

along with its rich heritage of in-car audio solutions, paired with Samsung's industry-leading mobile and consumer offerings. As we consider the autonomous future and our combined capabilities, we are poised to meet consumer demands and own even more of the passenger economy, the free time that will open up as systems become more self-controlled.

FUTURE POSSIBILITIES

No longer will the disparate elements of a car exist in isolation; they must be combined seamlessly in a streamlined interface, providing a safe, user-friendly, and cost-effective solution. A forward-looking vehicle system leverages technologies such as AI, AR, voice-based controls, intelligent connectivity, and cloud-based computing. HARMAN, alongside other leading tier-one providers, is focusing on humanizing the autonomous car through a digital cockpit and overall UX that is contextual, intuitive, and personalized.

As the shift toward the autonomous car progresses, the automotive industry, technology companies, and governments will need to work in tandem in the name of advancing technology, safety, and policy, while keeping consumer needs and interests at the forefront. Within the car, drivers-turned-riders will be able to do more in the same amount of time. The commute will be revolutionized: if you forget to schedule a meeting, you can request one via voice control; you could sit back and read your favorite novel; or you could even immerse yourself in a virtual reality game as an after-work stress reliever. OEMs will partner with automotive suppliers and cloud-based computing experts to create a vast suite of productivity and entertainment services, seamlessly connecting the car and the phone, the car and the home, or the car and the office.

Collaboration is vital, and no one company (or ecosystem) can do it all. Even so, UX/user interface is critical for automaker differentiation. Automakers' brands will live and die based on the unique, branded experiences they deliver inside the car cabin, making the battle for the passenger economy business even more competitive. The design of HMI systems is critical to the development of these branded experiences, with an intuitive display and interaction with data at the heart of the solution. This enables more seamless, natural interaction between the driver and vehicle, and it provides a chauffeur-like technology that will be a vital component of the car of the future.

ABOUT THE AUTHOR

Tim VanGoethem is the vice president of product and strategy at HARMAN.

Bits Versus Electrons *(continued from page 107)*

Today, surprisingly, JavaScript has become a high-performance programing environment thanks to clever ideas like dynamic compilation. Tools like TypeScript assist programmers by allowing them to provide hints to the integrated development environment. In addition, there is a whole set of tools to facilitate the sharing of packages. An event-driven single-threading approach removes much of the complexity of high-performance applications. With platforms such as NodeJS, JavaScript isn't just for the web. PWAs are just one class of applications written using the language and associated tools.

A PWA WORLD?

Not quite. While I am excited about PWAs and see many venues, such as TV v2, where they are vital, there is still a need for a variety of approaches to programing and software development. There will continue to be a need to program applications ahead of what is available to the PWA environment and a need for developers to create their own extensions.

There is still a need for a variety of approaches to programing and software development.

Google, Microsoft, and many others are embracing PWAs. For Microsoft, there is recognition that they can make money providing service using their Azure platform, and, for Amazon, their Amazon Web Services. For me, PWAs are exciting because they bring back some of the excitement of writing and sharing applications without all of the complexities of applications meant for wide market sales.

Today's PWAs are built on the current web, which is optimized for content distribution and commerce. As we explore new use cases, I expect to see much innovation, including the development of more peer technologies rather than a focus on delivering services. With that being said, the current technologies and protocols are already a strong basis for delivering these capabilities.

Consumer electronics devices will increasingly use PWAs either internally or as an interface. More than that, PWAs empower prosumers to wield software like they did soldering irons in the past.