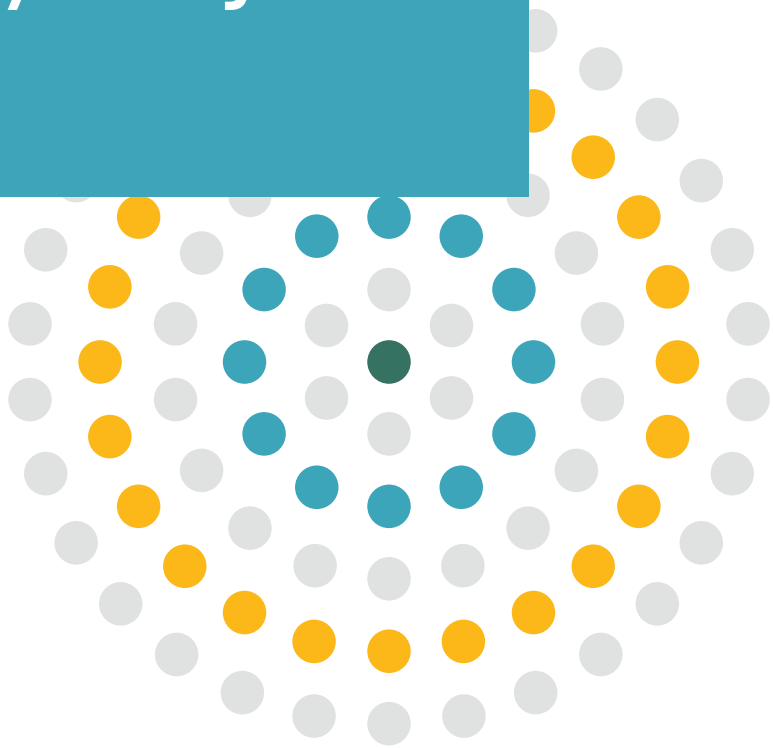
Finsemble

Considering PWAs for your Desktop Interoperability Project?

READ THIS FIRST.



Author: Terry Thorsen
Cosaic Co-founder and Chief Technology Officer

Cosaic

Introduction

In the world of desktop interoperability, there's currently a lot of buzz around Progressive Web Apps, or PWAs. PWAs are applications built from the web technologies we know and love, like HTML5, CSS, and JavaScript, but with a feel and functionality that rivals an actual native app. As developers of smart desktop solutions, Finsemble is right in the middle of the PWA conversation. PWAs present exciting possibilities, and may well be an important part of desktop interop in the future, but right now, they are not a replacement for Electron. This article will cut through the marketing noise around PWAs to show what's great, where they fall short, what you can do with them now, and what the future may hold.

The great news is that you don't need to put your desktop interoperability (interop) project on hold waiting for PWA technology to mature. You just need to work with a vendor that has a solution for now, a vision for later, and a plan to get you there.

What is desktop interoperability?

Before diving into the gritty details, it's important to understand how desktop interop works and what parts of the puzzle PWAs hope to solve. Desktop interop is a new technology that allows independent apps to be combined into a seamless user experience. Imagine integrating new apps with existing solutions in hours instead of months. That's desktop-interop.

Desktop interop functionality comes in two parts:

1. Things a browser can do
2. Things a browser cannot do

To do the things a browser cannot do you need a "runtime".
The big things a runtime bring to the table are:

1. Communication between cross-domain/cross-application windows (at a minimum, a message bus, ideally a workflow engine, FDC3 support, etc.)
2. Management of multiple windows on screen (e.g. snap & dock, swimlanes, tabbing, etc).

Here's the catch: the runtime must be installed on your users desktop, and that can be a significant barrier to adoption. Which leads us to PWAs.

NOTE FOR DEVELOPERS

Runtimes contain several pieces of technology:

1. Code for displaying the web content - All vendors use Chromium. This can be done either through Electron or CEF (chromium embedded framework). Note that support for web content running inside a browser tab (or, as we discuss below, in a PWA) is also a critical capability often overlooked by interop vendors.
2. Window management code—This is usually Javascript running in Electron but can also be proprietary native Win32 code. To support native apps (and not all vendors do!) you always need some proprietary code.
3. A message bus - This can be Electron's built-in bus, but is usually proprietary because Electron can't handle native apps.

Short History of PWAs

If you say "PWA" to most people in Silicon Valley they immediately think about phones. PWA has mostly been an approach to mobile solutions, rooted in Steve Jobs' vision that all iPhone apps be written in HTML5 (See History of PWA article here). When the iPhone came out, Safari was limited, HTML5 was too new, and iPhones weren't powerful enough, but these days a lot has changed. Chrome is good. HTML5 is mature. Phones are powerful. So PWAs time has finally come—on phones.

On a phone, an HTML5 page that works like an app is pretty easy to imagine. Just add an icon to your phone's home screen that pulls up a url. When you click the icon, open up a browser page with the location bar hidden, so it doesn't feel like you're in a browser. Let the page save data locally on your phone, and you have a PWA. That's really all there is to the current generation of PWAs.

PWAs can work on a desktop—in exactly the same way. You get an icon. You get a webpage without a location bar. You get the ability to save data locally. A webapp with an icon and a "chromeless" window is a big deal (and in fact I discussed this in my Covalence presentation last year) but it isn't Desktop Interop. It's just "Desktop".

Right now, you can cobble together a DIY desktop interop using the browser but it will be severely limited. It can pop up windows but not manage them. Forget about fancy UI like custom title bars and slim, dockable toolbars, not to mention the real deal killers like native apps and process management. Basically, PWAs today can do what browsers have been doing for the past five years, only with a hidden location bar and a desktop icon.

Where does PWA fit into the desktop interop conversation?

PWAs can potentially play two roles in desktop interop:

> **A way to display a web application that needs to participate in desktop interop, just like an Electron window, a browser tab, or even a WebView inside a native application.**
> Or;
> **A replacement for the core runtime itself.**

In the first case, PWAs are "just another app" running in your smart desktop, much like .NET, Java, and Electron windows. It's critical to evaluate your interop vendor for their ability to manage PWAs as apps. Not all desktop interop solutions can snap and dock a .NET window to an Electron window, and not all of them treat PWAs as first class citizens in the runtime.

The second role is exciting because it positions PWAs as a replacement for the entire runtime. If you didn't need a runtime then you wouldn't have to install anything on a user's desktop, lowering deployment costs and hassle. PWAs used in this way would also solve many security concerns, since they rely on Chrome, a ubiquitous and well-trusted platform.

The problem is that PWAs don't do most of what a runtime does (at least not yet) and depending on how PWAs evolve, they may never be able to.

PWAs Moving Forward

There are some people in Silicon Valley who are really interested in making PWAs do more on the desktop:

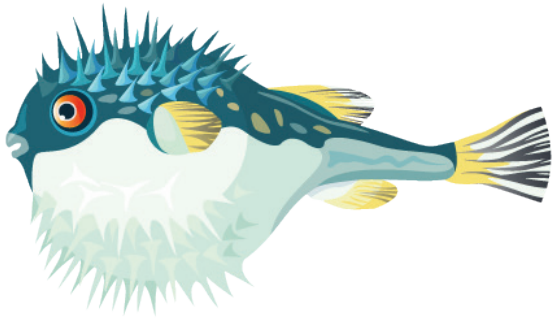
- **Electron Developers** People who build Electron apps would be stoked if their customers didn't need an installer. These include companies like Slack, Spotify and thousands of individual developers.
- **Google** Google's ChromeOS only runs browser apps. PWAs are a technology that can make that operating system more competitive. Google is, of course, the primary contributor to the Chromium project.
- **Microsoft** Microsoft is "all in" on JavaScript and web development. They see JS/Web as the future of desktop development. Microsoft owns Github, which manages the Electron project and has decided to use Chromium as the core browser engine in Windows.
- **Chromium Team** They want Chrome users to get a desktop experience but be able to t rust that Chrome is protecting their security. Both Google and Microsoft engineers contribute to Chromium.

The interest in PWAs on the desktop has spawned a new Chromium project called "FUGU". This project is intended to add more native capabilities to Chromium (it's called "fugu" because of the fish—giving desktop capabilities to the browser is dangerous. Like cutting and eating the poisonous fugu fish, it has to be done just right).

Project FUGU offers APIs for some things that can currently only be done with Electron. Most notable for Desktop Interop is FUGU's "window placement" API. However, as powerful as that API is, it's still missing important features that Electron provides like BrowserView, transparent windows, floating toolbars, and intra-window security.

Project FUGU and Desktop Interop

It is unclear to what degree project FUGU will make PWAs practical as the runtime replacement for desktop interop. The security concerns of the FUGU team are different than those of enterprises who manage their own software and know their users. FUGU is concerned about malicious developers who might build Electron based browsers that steal payment information, code written by random developers and downloaded by random users.



Because of these concerns, the FUGU team may have to make concessions that would hurt desktop interop in the enterprise. Two examples are "frameless windows" and "customizable titlebars". These two technologies are fundamental to the desktop interop experience, but they are at odds with FUGU's security mandate. End users trust Chrome because there are parts of Chrome's UI that display security information, like the "lock" that tells you a site is secure. Everybody knows those areas can't be overridden by a website and so everybody knows that Chrome is protecting them. But frameless windows and customizable toolbars eliminate those areas in order to gain a cleaner user experience. FUGU might not be able to support that.

It's possible that the FUGU project team will be able to find a middle ground, but those types of issues will determine how much FUGU changes the game for desktop interop. Our belief is that project FUGU will certainly have an impact. Likely, it will simplify the runtime stack but not entirely eliminate it. It will make "falling back" to a browser a more acceptable experience.

What we do know for certain is that project FUGU will not address legacy native applications. Any desktop interop solution that includes native applications will always need to have a runtime installed on the system.

What Decision to Make?

We feel that Electron is the right solution for right now. That's why our runtime is based on Electron—it gives us a lot right off the shelf. Our endorsement for using Electron comes with one caveat—once you are working in a multi-application environment on the desktop you want to be sure Electron is secured (in fact, we recently contributed a FINOS-hosted open-source project called Secure Electron Adapter used for this purpose). As an interop vendor committed to making the future possible today, while leveraging your technology investments of the past, we treat PWAs as first class citizens inside Finsemble—Electron windows, NET, Java, Browser tabs, PWAs, etc—all work together, providing the flexibility you need now, and a capability not all vendors can support.

As the power of PWA technology increases we will evaluate whether to incorporate it into our runtime, or to provide a standalone browser-only runtime. If it does become possible to eliminate runtimes entirely then everybody will be happy.

Desktop interop software will be on your desktop for 10-15 years or longer, which means we're building for the long haul. Technological change within that time frame is a certainty.

We advise you not to wait for next generation technology to emerge before rolling out a desktop interop solution. Work with the technology that is available, and pick a vendor who you feel will support you across the next decade or more with the best feature set today, the right vision for the future, and a roadmap to get you there.

About the Author
Terry Thorsen, Cosaic Co-founder and Chief Technology Officer

In 1994, Terry wrote the first online trading system which later became Ameritrade. He then went on to found Automated Financial Systems, a provider of financial software to banks and brokerages.