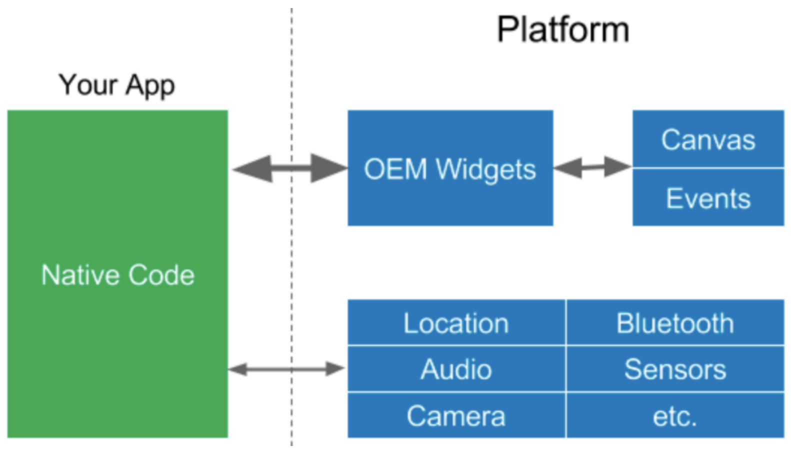
The Evolution of Mobile App Development

Jaideep Ganguly, Head Compass India Development Center, Hyderabad, India

1. Introduction

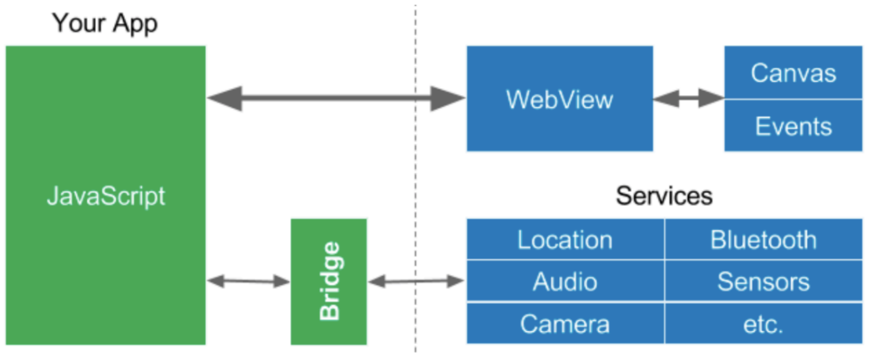
Mobile app development is lots of fun as one gets the opportunity to leverage hardware capabilities such as GPS, camera, maps, sensors and so on to build really cool apps and delight customers who are on the move!. At Compass we are pushing the envelope and building some cool features that work in tandem with great server capabilities and machine learning algorithms.

In this article, we will trace the tech evolution in mobile development, explain the nature of the stacks to cut the clutter as it is a jungle out there. Mobile application development is just over a decade old and is rapidly evolving. The Apple iOS SDK was released in 2008 and the Google Android SDK in 2009. These two SDKs were based on Objective-C and Java respectively. The app talks to the platform to create widgets, or access services such as camera. The widgets are rendered to a screen canvas and events are passed back to the widgets. The architecture is simple with the drawback that one has to create separate apps for each platform because the widgets and the native languages are different. Android apps can now be written in Kotlin which makes the code succinct due to its support of the functional style.



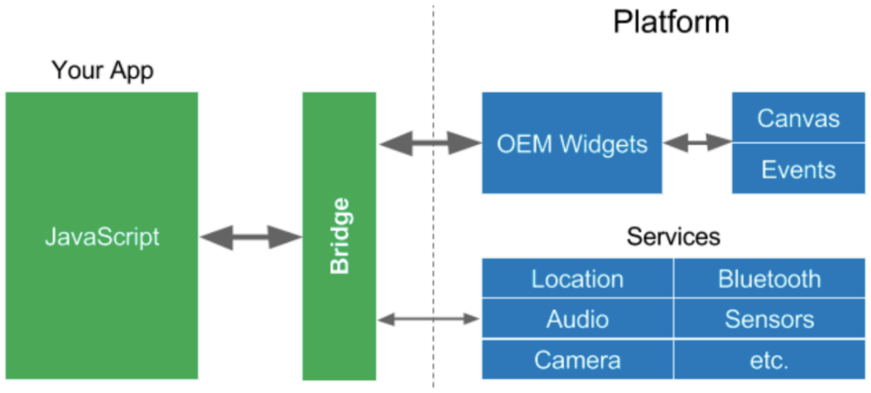
2. JavaScript & WebViews

Not surprisingly, JavaScript and WebViews were used to build the first cross-platform frameworks such as Apache Cordova, PhoneGap, Ionic, etc.



But it is difficult for JavaScript to talk directly to native code (like the services) so they go through a “bridge” that context switches between the realms of JavaScript realm and native. Since platform services are typically not invoked frequently, it did not affect performance significantly.

Reactive web frameworks such as ReactJS have become popular as they simplify the creation of web views through the use of programming patterns borrowed from reactive programming whose goals are that applications should be *responsive*. This implies that the applications need to be *scalable* and *resilient*. To achieve that, a *message driven* architecture is used as the foundation. React Native was launched in 2015 to bring forward the many benefits of reactive-style views to mobile apps.



3. React Native

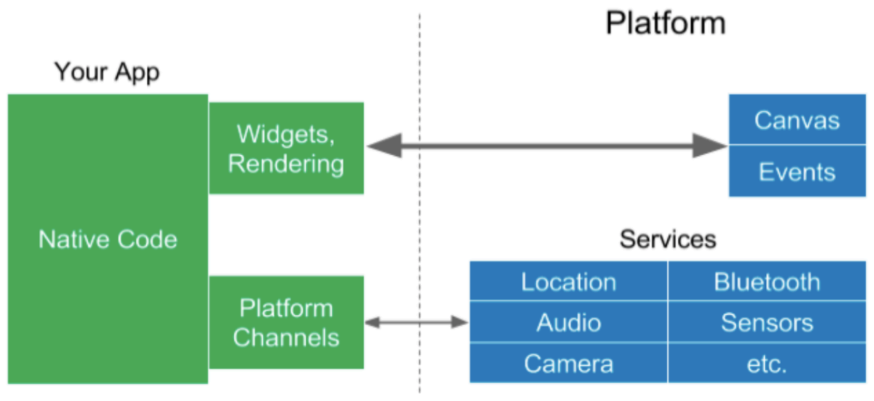
React Native is very popular (and deserves to be), but because the JavaScript realm accesses the platform widgets in the native realm, it has to go through the bridge for those as well. Widgets are typically accessed quite frequently (up to 60 times a second during animations, transitions, or when the user “swipes” something on the screen with their finger) so this can cause performance problems. While each realm is fast, the bottleneck occurs when one transitions from one realm to another and that means we need to minimize the transitions to keep the performance satisfactory.

4. Flutter & Dart

Enter Flutter which takes a different approach to avoiding performance problems caused by the need for a JavaScript bridge by using a compiled programming language, namely Dart. Dart is compiled “ahead of time” (AOT) into native code for multiple platforms. This allows Flutter to communicate with the platform without going through a JavaScript bridge that does a context switch. Compiling to native code also improves app startup times. Flutter is the only mobile SDK that provides reactive views without requiring a

JavaScript bridge.

Flutter has a revolutionary approach in implementing its widgets. Widgets are the elements that affect and control the view and interface to an app. The look and feel of widgets is paramount, they need to look good, work on various screen sizes, feel natural, be fast and should be extensible and customizable. Flutter does not use the platform widgets (or DOM WebViews), it provides its own widgets.



Flutter incorporates the widgets and renderer from the platform into the app and this allows them to be customizable and extensible. Flutter simply requires the platform's canvas to render the widgets for them to appear on the device screen and access to events (touches, timers, etc.) and services (location, camera, etc.). There is still an interface between the Dart program and the native platform code that does data encoding and decoding but this is orders of magnitude faster than a JavaScript bridge! Moving the widgets and the renderer into the app does affect the size of the app. The minimum size of a Flutter app on Android is about 5 MB, which is similar to minimal apps built with comparable tools.

One of the biggest improvements in Flutter is how it implements layout. Layout determines the size and position of widgets based on a set of rules or constraints. Traditionally, layout uses a large set of rules that can be applied to any widget. The rules implement multiple layout methods and can be very complex. This makes layout slow. Even worse, layout performance is typically of order N-squared, so as the number of elements increases, layout slows down even more.

However, most layouts are pretty simple, text on a scrolling page, fixed rectangles whose size and position depend only on the size of the display, tables, floating elements, etc. Since most layout is local to a subtree of widgets, and that subtree typically uses one layout model, only a small number of rules need to be supported by those widgets. So, instead of having a large set of layout rules that could be applied to any widget, each widget would specify its own simple layout model. Because each widget has a much smaller set of layout rules to consider, layout can be optimized heavily.

Furthermore, to simplify layout even further, Flutter turned almost everything into a widget. In Flutter, centering and padding are widgets. Themes are widgets, which apply to their children. And even applications and navigation are widgets. Layout in Flutter is so fast it can be used for scrolling! Scrolling must be so instantaneous and smooth that the user feels like the screen image is attached to their finger as they drag it across the physical screen.

In summary, building apps in Flutter and Dart is simple and fun!