Writing a Hybrid Mobile Application with PhoneGap and the Dojo Toolkit

How to write a Hybrid Mobile Application, where part of the application is installed on a mobile device and part is fetched from the web

Andy Dingsor
Advisory Software Engineer
IBM, Research Triangle Park, NC

Introduction

Deciding how to deliver content to mobile users is a large challenge for application design architects. Applications must be efficient to update, must look like native mobile apps, and should leverage familiar technologies. In this article, web designers learn how to design hybrid mobile and web-based applications with PhoneGap and the Dojo Toolkit, using HTML, CSS and javascript technologies.

Application Design Problems

Pure mobile apps have all their content installed on a mobile device. Pure web-based apps deliver all their content from a server. These represent opposite extremes which each have their own advantages and disadvantages.

Pure mobile apps are easy for the user to find, and fast and easy to start. However, because all content is stored on the mobile device, it can be hard to deploy updates with fixes and enhancements. In addition, each update to an app store may be subject to a review and approval process over which you have no control of schedule or criteria.

Pure web-based apps are hard for the user to find and start. The user must remember to start a browser rather than a specific app, then must remember a URL, and either type it or deal with bookmarks. On the other hand, web-based apps can be updated instantly for all users because the content is hosted on a server.

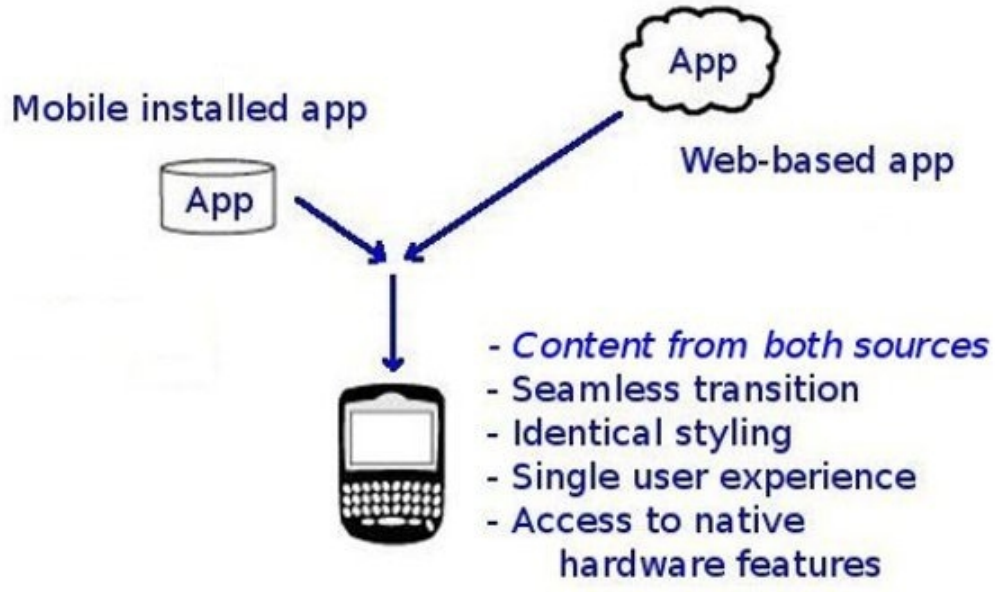
The Hybrid Application Solution

Combining the best features of mobile and web-based apps provides a powerful design pattern. A "hybrid app" is an application which has been partitioned so that part runs on the mobile device and part is delivered by a server. The mobile app can be easily found and started quickly by the user. Keeping the mobile app small reduces the risk for

updates. And placing most content in the web-based app allows instant updates for fixes and enhancements, simply by modifying the web server content.

Hybrid apps deliver the best of both worlds.

Figure 1. Hybrid Application Solution



Requirements

There are several requirements to make a hybrid app successful.

* A hybrid app must make a seamless transition between the mobile and web-based content. Both the mobile and web-based content must be presented within the same environment, and the user should not be able to detect whether the content is coming from the mobile device or from the web.

* The user interface must look like a mobile app on both the mobile and web-based sides. All buttons, widgets, controls, and styling must look and feel the same regardless of whether the content is from the mobile device or from the web.

* Content from both the installed mobile app and the web-based app must have access to native hardware features of the mobile device, such as the camera and GPS.

To meet these requirements, the hybrid app developed for this article used two open-source packages: PhoneGap and the Dojo Toolkit.

What is PhoneGap?

PhoneGap is an open-source mobile framework that presently supports six platforms (iOS, Android, Blackberry, etc). It enables authoring of native mobile applications using well-known HTML, CSS, and javascript languages, it provides access to native hardware features of the mobile device through a common platform-independent javascript API (camera, GPS, etc), and it packages your app so it can be published in the various app stores and markets.

The app developed in this article makes use of the following PhoneGap features:

* The PhoneGap website provides instructions for setting up platform-specific build environments consisting of Eclipse, each platform SDK, and PhoneGap to build native mobile apps which are ready to install and publish.

* A PhoneGap plug-in, named ChildBrowser, is used to demonstrate a seamless transition from mobile-based HTML content to web-based HTML content within the app environment. The ChildBrowser plug-in displays an external web page within the PhoneGap application, rather then spawning a separate browser.

* The PhoneGap javascript APIs are used to demonstrate that both mobile and web-based content can access native hardware information about the mobile device.

Support for PhoneGap is available through the open-source community. While IBM developers contribute to the PhoneGap project, IBM does not ship or support PhoneGap.

What is the Dojo Toolkit?

The Dojo Toolkit is an open-source javascript library designed to speed development of browser-based applications. It includes a huge collection of widgets and controls, it hides and accommodates many differences between various browsers, and includes a wide variety of utility classes and functions for miscellaneous tasks such as XHR. It provides built-in styles for several popular mobile platforms, it runs in a clean namespace to avoid collisions, and provides a process to develop custom optimized builds to save space and time.

The app developed in this article makes use of the following Dojo features:

* Lists and buttons from dojox.mobile are used in both the mobile and web-based parts of the app. This demonstrates that the same controls can be used on both sides.

* The built-in CSS styling for Android is also used in both the mobile and web-based pieces. This demonstrates how to deliver the identical look-and-feel on both sides.

Dojo is included in the IBM WebSphere Application Server Feature Pack for Web 2.0 and Mobile v1.1 and may be packaged as part of a production app. To trade simplicity for performance, the app developed in this article downloads Dojo dynamically from the Google CDN.

Getting started

A small hybrid app is presented in this tutorial. It consists of two pieces, one installed in a mobile device, the other delivered from a server.

The core content of the mobile piece is one HTML file, index.html. This HTML file uses PhoneGap and Dojo to present the appearance of a native mobile app, to access hardware features on the mobile device, and to make a transition to the server piece. Its details are explained in the next section.

The core of the server piece consists of another single HTML file, serverapp.html. This HTML file also uses PhoneGap and Dojo to present the same mobile appearance, and to access the same hardware features on the mobile device. Its details are explained in a subsequent section.

Two JPG splash images are included to add a little bling. One is used in the mobile app, the other at the web-based app. The server image is used to highlight the normally invisible transition from mobile app to web-based app for demo purposes. Don't be confused; a server splash image is not recommended for production use.

Figure 2. Splash images

Developing the mobile app

The Android platform was chosen for this article. Instructions from the PhoneGap site were used to set up an Android build environment consisting of Eclipse, the Android SDK and simulator, and PhoneGap: http://www.phonegap.com/start#android

To make a seamless transition from the mobile app to the server app, a PhoneGap plug-in named ChildBrowser was added according to these instructions: http://simonmacdonald.blogspot.com/2011/05/installing-childbrowser-plugin-on.html

ChildBrowser consists of a java class and javascript files, and a tweak to a manifest file. This key plug-in allows the external web page to load within the PhoneGap application, rather than starting a new browser.

The Dojo Toolkit was chosen to make the app look and feel like a native Android app. A few classes from dojox.mobile, including a view, list, and button, were used to give a sampling of what is possible. For simplicity in this demo, the android.css from Dojo was used directly, and Dojo was downloaded from the Google CDN.

Two files were created for the mobile piece: an HTML file and a splash image. (Complete source can be downloaded here).

Mobile index.html

This file is located in the eclipse project workspace .../HelloHybrid/assets/www/.

Both Dojo and PhoneGap statements are located in the heading section. The Dojo statements gain access to the Dojo library from the Google CDN, and list the required dojo classes.

Listing 1. Dojo Mobile Requires

```
<!-- DojoMobile -->
<link href="http://ajax.googleapis.com/ajax/libs/dojo/1.6/
   dojox/mobile/themes/android/android.css" rel="stylesheet" />
<script type="text/javascript"
   src="http://ajax.googleapis.com/ajax/libs/dojo/1.6/dojo/dojo.xd.js"
   djConfig="isDebug:true,parseOnLoad:true">
</script>
<script type="text/javascript">
   dojo.require("dojox.mobile.parser");
   dojo.require("dojox.mobile");
   dojo.requireIf(!dojo.isWebKit,"dojox.mobile.compat");
</script>
```

The PhoneGap statements reference javascript files for PhoneGap and ChildBrowser.

Listing 2. PhoneGap References

```
<!-- PhoneGap -->
<script type="text/javascript" charset="utf-8" src="phonegap.0.9.5.js"></script>
<script type="text/javascript" charset="utf-8" src="childbrowser.js"></script>
```

The body section presents a heading and a list to the user. The list includes the splash image and a link to the next view.

Listing 3. Heading and List

```
<div id="mainView" dojoType="dojox.mobile.View" selected="true">
      <h1 dojoType="dojox.mobile.Heading">HelloHybrid Mobile v0.5</h1>
       <ul dojoType="dojox.mobile.EdgeToEdgeList">
         <li>
            <img src="pix/hhm.splash.jpg">
         </li>
         <li dojoType="dojox.mobile.ListItem" moveTo="welcomeView">
            Welcome
         </li>
      </ul>
  </div>
```

The next view contains two buttons. One button fetches mobile device-specific information using the PhoneGap native-device API.

Listing 4. Button to fetch device specific info

```
  <div id="welcomeView" dojoType="dojox.mobile.View">
     <h1 dojoType="dojox.mobile.Heading"
        back="Main"
        moveTo="mainView">
        HelloHybrid Mobile
     </h1>
     <p>Click for local mobile device info:</p>
     <button dojoType="dojox.mobile.Button"
        onClick="var deviceInfoID =
           dojo.byId('deviceInfo'); deviceInfoID.innerHTML =
                    'Device Name: '     + device.name     + '<br />' +
                    'Device PhoneGap: ' + device.phonegap + '<br />' +
                    'Device Platform: ' + device.platform + '<br />' +
                    'Device UUID: '     + device.uuid     + '<br />' +
                    'Device Version: '  + device.version  + '<br />';">
        Device Info
     </button>
     <p><div id="deviceInfo"></div></p>
```

The other button makes the transition to the web-based app using the PhoneGap ChildBrowser. (Please note the link in this button is hard-coded to a server with a private hostname for this demo. A production app should point to a real server page.)

Listing 5. Button to transition to web-based app

```
     <p>Click to access server application:</p>
     <button dojoType="dojox.mobile.Button"
        onClick="window.plugins.childBrowser.showWebPage(
           'http://my.test.server.com/html/serverapp.html','true');">
```

```
    Server App
</button>
```

Mobile splash image hhm.splash.jpg

This image file is located in the eclipse project workspace under:
.../HelloHybrid/assets/www/pix/

This completes development for the mobile side. An installable APK file for Android can be built using the Eclipse, PhoneGap, and Android SDK environment, and then be installed on an Android phone. To install it, the APK can be emailed to a Google Mail address, and the gmail app on the phone presents an 'install' button for the received APK.

Developing the web-based app

A new HTML file and image file were also created for the server-side (complete source code can be downloaded here).

Web-based serverapp.html

For this demonstration, a static HTML file is delivered by an Apache web server. In production, web-based content may be delivered from static files, dynamically from a J2EE app server, or various other technologies.

Apache serves HTML files from a directory listed in its configuration file (eg, httpd.conf). The default directory on linux systems is typically /var/www/html. For this demo, the serverapp.html file is located in the default HTML directory /var/www/html/.

To begin creating the serverapp.html, the mobile index.html file was initially copied into serverapp.html in order to preserve most of the PhoneGap and Dojo contents. The PhoneGap ChildBrowser content and the Dojo "Server App" button were removed, since they were not necessary on the server side. The splash image link was also changed to reference the server splash image.

Web-based splash image hhs.splash.jpg

This image file is located in an image directory served by the Apache server:
/var/www/html/pix/.

This completes development for the server side. The HTML file and splash image are being served by the Apache server, and can be fetched by the mobile app. All the pieces are now in place.

Demonstration

The hybrid app is ready for a live demonstration, now that the mobile piece has been built and installed on a phone and the web-based piece is being served by an Apache server.

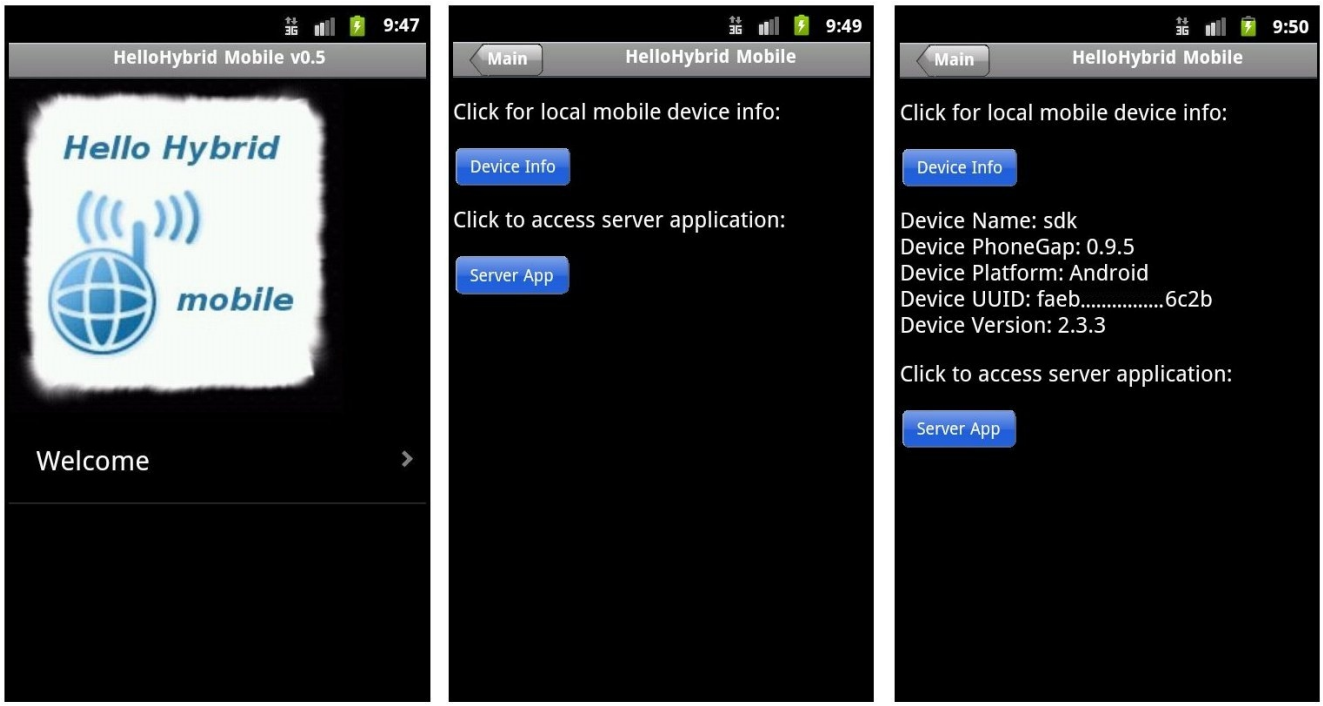
The mobile app is started, the user clicks through all the views and buttons, and the following screen captures are observed...

The screen captures in Figure 3 shows the mobile-based content. The content for all screens in this figure (image and HTML) is packaged and installed in the mobile app.

The mobile splash image is presented when the mobile app starts. Clicking "Welcome" moves to the next view. Clicking "Device Info" fetches information about the mobile device through the PhoneGap API. This is the same API used to access native hardware features such as the camera and GPS. Clicking "Server App" transitions to the web-based app through the PhoneGap ChildBrowser.

This demonstrates three key design requirements for the mobile piece: the Dojo Toolkit provides widgets and styling which look like a native Android app, the PhoneGap API allows access to native hardware features of the mobile device, and the PhoneGap ChildBrowser provides a seamless transition to the web-based app.

Figure 3. Content from mobile installed app.



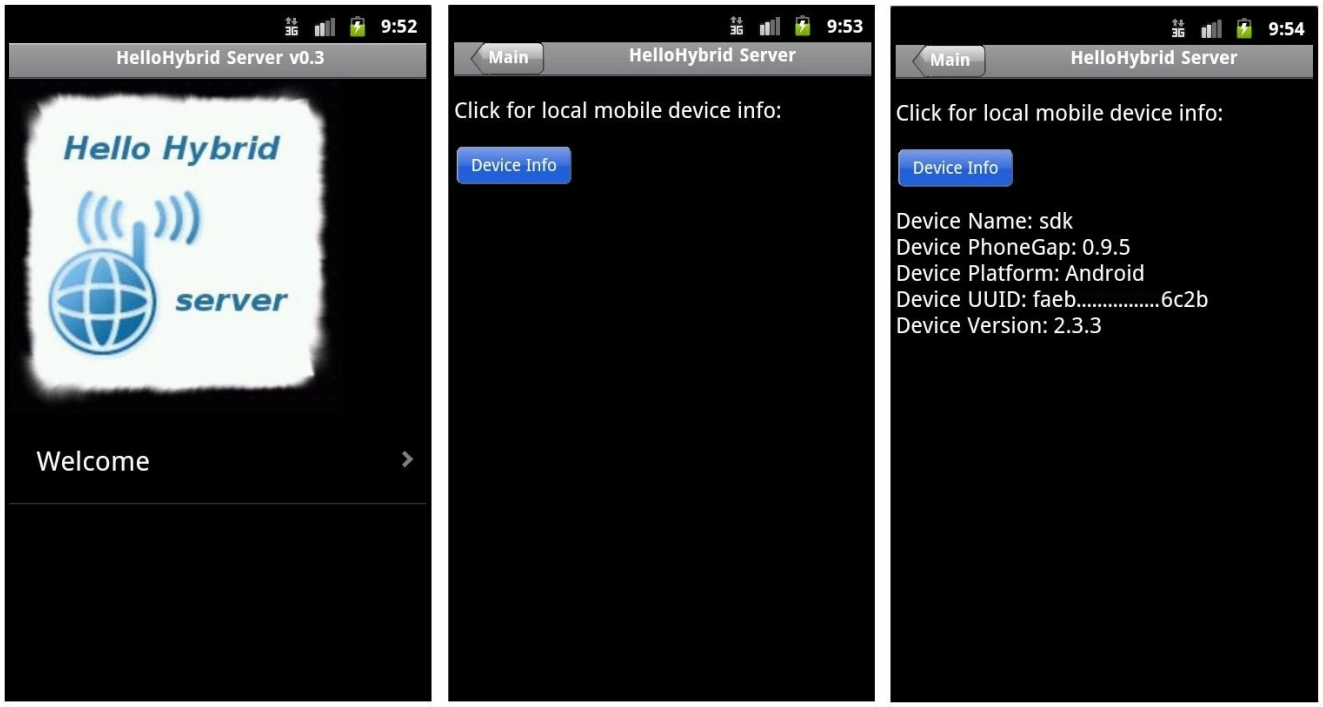
Clicking the “Server App” button makes the seamless transition from the mobile content to the web-based content.

The screen captures in Figure 4 show web-based content. The content for all screens in this figure (image and HTML) is delivered from the server.

The server splash image is presented after the transition from mobile to web-based app. Clicking "Welcome" moves to the next view. Clicking "Device Info" fetches information about the mobile device through the PhoneGap API.

This demonstrates the remaining design requirements for the web-based piece: it shows that Dojo presents an identical native-styled user experience from both mobile and web-based apps, and it demonstrates that a web-based app has access to the native hardware features of the mobile device via PhoneGap, just as it did in the mobile-installed piece.

Figure 4. Content from web-based app



Conclusions and Recommendations

This article explained how hybrid apps deliver the best features of pure mobile apps and pure web-based apps. You learned how to design hybrid mobile and web-based applications with PhoneGap and the Dojo Toolkit, using HTML, CSS and javascript technologies:

* Create a small mobile app which is easily found, starts fast, and seamlessly transitions to the web-based app. Place the bulk of the content in the web-based app. This allows the bulk of the content to be updated instantly on the server.

* Use PhoneGap to create installable mobile apps. PhoneGap helps build installables ready to be uploaded to app stores and markets.

* Use PhoneGap ChildBrowser to make a seamless transition from mobile app to web-based app. This hides the transition from the user.

* Use the Dojo Toolkit to style apps identically and natively on both mobile and web-based sides. This provides the user with a familiar native-looking experience on all screens.

* Use PhoneGap to access native hardware features of the mobile device from both the mobile app and web-based app. This allows your application to access geolocation data, camera images, and other mobile device features.

PhoneGap and Dojo together provide a powerful environment to deliver hybrid apps.

Attachments

- Download sample files here.

References

- Use PhoneGap.
- PhoneGap at Wikipedia.
- PhoneGap development sandbox and 'Hello World' instructions.
- PhoneGap ChildBrowser.
- Dojo Toolkit.
- Dojo Toolkit at Wikipedia.
- Dojo optimized builds (not used in this tutorial).

Further advice

* To become familiar with PhoneGap, follow the tutorial to create a "hello world" program (http://www.phonegap.com/start#android). The app can be deployed and run on both an Android simulator and on a physical Android device.

* Consider where to package the Dojo Toolkit. The app developed in this article references Dojo from the Google CDN, which is simple but can be slow to load. A production app can benefit from including Dojo within the app. An optimized Dojo build contains only the content required by the app in order to improve speed and efficiency.

* Consider all mobile platforms. The demo app was written to run on Android only. Production apps should run on additional platforms, and can exploit the CSS and utilities built into Dojo to ease the challenge.

* Consider national language support. Many Dojo widgets include national language support which simplifies internationalization.

Acknowledgements

Many thanks to IBM employee Bryce Curtis for his contributions to this article. Thanks also to IBM employee Todd Kaplinger for his encouragement to use PhoneGap.

About the author

Andy Dingsor is an advisory software engineer at IBM. He has developed a wide variety of hardware and software products, including a video conferencing system, PCMCIA Wireless WAN adapters, a TCP/IP network load balancer, and the WebSphere Application Server. Andy holds seven U.S. patents on FM radio modulation, networking, and software techniques. He occasionally commutes to work by bicycle.