Argos: Building a Web-Centric Application Platform on Top of Android

Rich Gossweiler, Colin McDonough, James Lin, and Roy Want

How do you combine the strengths of the Web with the native capabilities of the phone? Anyone who has learned how to write a native mobile application knows that it's not straightforward. Compared to Web applications, native applications don't integrate as naturally with the richness of the services and content that the Web provides. If developers could apply their knowledge of designing Web applications to smartphone application design, they could leverage years of experience to rapidly accelerate phone application development. How do we build a mobile application development platform that supports the fluidity, richness, and power of the Web combined with the special capabilities of mobile devices?

A Web-centric approach to mobile applications supports

- rapid prototyping for mobile applications,
- easy, unified integration with Web services, and
- access to the mobile hardware resources through JavaScript.

Platforms and standards such as PhoneGap (www.phonegap.com), WebOS,[1] and HTML5[2] address these points to varying degrees. Argos is a Web-centric phone development platform designed to expose the Android operating system's rich set of features. Developers write HTML and cascading stylesheets (CSS) along with Argos's JavaScript library to gain access to extended features. Examples include telephony, haptic feedback, Bluetooth, Near Field Communication (NFC), and even graphics rendering using multiple overlays.

In this article, we describe this approach, introduce the Argos system (see Figure 1), compare Argos with related platforms, detail its architecture, and discuss current conclusions and future work.

THE CASE FOR MOBILE WEB APPS

Today, smartphone application development environments are reminiscent of the desktop environment of the 1980s. At that time, desktop developers used specialized integrated development environments (IDEs) to create executables that were sold as shrink-wrapped software or, later, made available for download over the Internet.

By the mid-1990s, the introduction of the Web and HTML made it much easier for people of all professions to write and share documents.[3] With the introduction of more powerful client-server models and embedded languages such as JavaScript with document object model (DOM) manipulation, these documents turned into applications. Millions of applications are now at the fingertips of anybody with an Internet connection.

Today, in the nascent world of mobile computing, we find ourselves back to using specialized IDEs to write standalone native applications, which are vetted and hosted in corporate marketplaces, such as those for Amazon, Android, and iOS. But because smartphones are nearly always connected to the Internet, we're missing out on all of the advantages gained from moving desktop-centric computing to Web-based applications. Three main issues surround the transition to Web tools for mobile application development: rapid development, Web services, and access to native phone resources.

Rapid Development

It's easier to craft innovative mobile user experiences using powerful HTML layout and rendering engines than current native IDEs. Web-based data-exchange protocols, multimedia support, and Web service APIs give developers universally accepted mechanisms for integrating millions of utilities and content elements. Toward this end, new protocols by the World Wide Web Consortium (W3C) enable richer Web mobile development, such as the detection of device orientation and access to platform resources. However, Web standards, which need international agreement, will always lag behind the new capabilities that phone manufacturers introduce.

Web Services

Smartphones have less CPU performance and less storage capacity than desktop PCs and network servers, but their performance is more than enough for thin clients. Thus mobile devices can access a wealth of Web applications through their browsers and can display both online and local data including real-time media. Further, the promise of higher-bandwidth ubiquitous 4G wireless connectivity will make an even stronger case for Web services in the future. Such services also have the flexibility to be combined in mash-ups, providing greater dynamic capabilities and extensibility.

Access to Native Phone Resources

Modern smartphones go beyond traditional computing architectures and include unique resources such as physical sensors (for example, an accelerometer and compass), GPS hardware, Bluetooth, and NFC. This makes possible the creation of exciting new applications such as context-aware software and more seamless financial services.

However, to access these resources you must create applications using native tools. For Android, this requires learning the details of its unique programming model, such as activities, intents, manifests, layouts, threading, and resources. It's powerful but cumbersome.

The Web programming model is a promising alternative that's easier to use, but the phone's browser can't access all of these native resources. We need a system that enables rich mobile Web development through easy-to-use libraries that keep up with the latest smartphone developments.

MOTIVATION FOR ARGOS

Argos was designed to address these three issues and focus on building a Web-centric development platform on top of Android. It aims to make it easier to rapidly develop Android applications, integrate the Web with Android development, and expose Android's rich set of native resources as APIs for Web-centric applications.

By focusing on Android, Argos attempts to bring out the strengths of the Android operating system at the cost of not supporting the other systems. Examples include support for voice recognition, NFC, and the ability to combine webpages, 3D graphics, and camera previews. Argos can also quickly introduce APIs in the future to track the latest technologies.

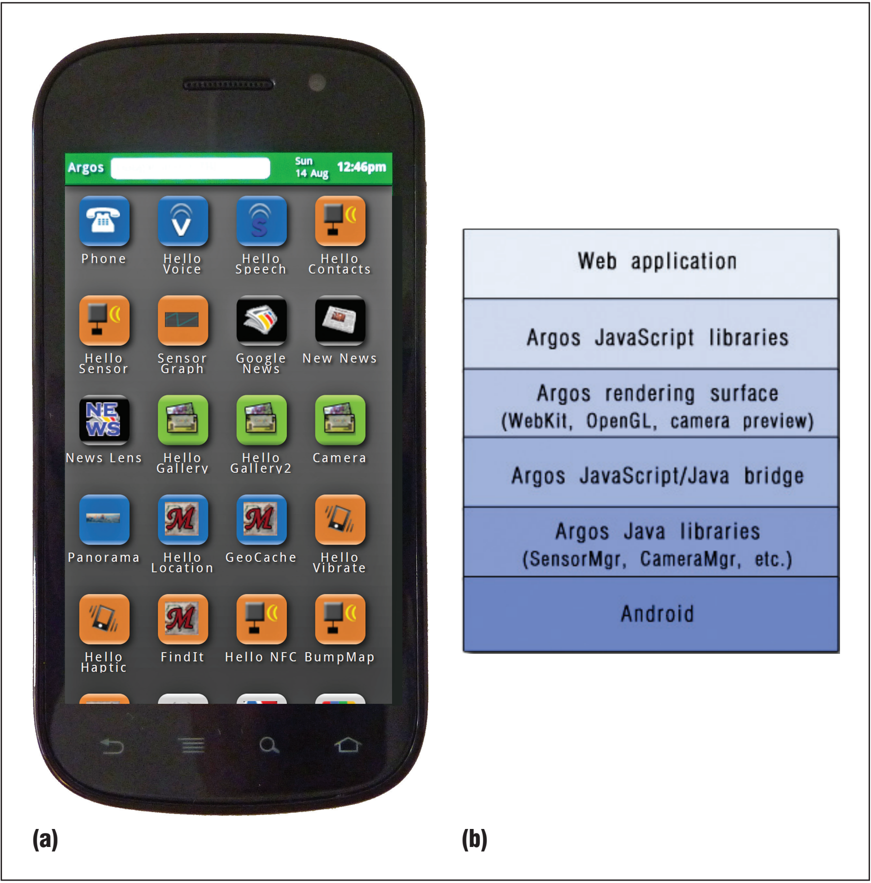


Figure 1. The Argos system: (a) an Argos launcher page and (b) integration of the Argos software stack with Android.

RELATED SYSTEMS

Several Web-centric mobile development platforms support various mobile operating systems to varying degrees. At one end of the spectrum are Web-centric operating systems such as HP's WebOS; at the other end are operating system-agnostic representations, such as HTML5. Popular platforms such as PhoneGap occupy the middle ground, not tuned to a specific platform, nor as generic as HTML5.

WebOS

WebOS is a Web-centric mobile operating system running on the Linux kernel. Developers write HTML/JavaScript applications that have access to the operating system's capabilities. Argos differs from WebOS in several ways. Argos is separate from the underlying Android operating system, so assumptions about the security model, the launching, threading, and resource management can be evolved without changing the operating system. Argos targets Android to leverage its specific

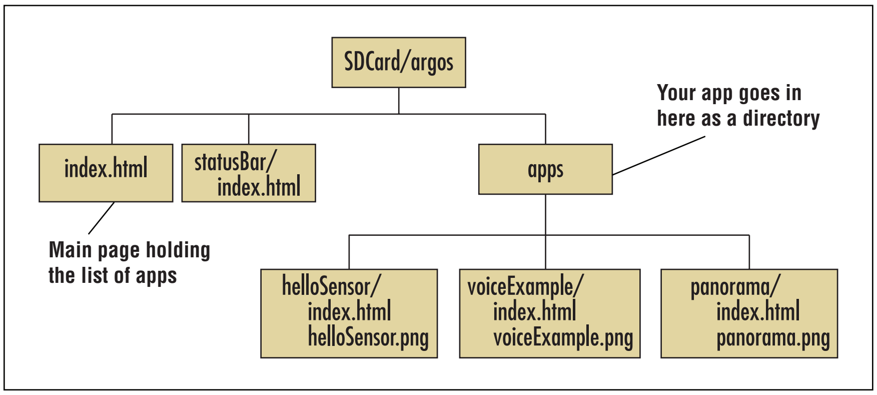

SDCard/argos
index.html
statusBar/ index.html
apps
Your app goes in here as a directory
Main page holding the list of apps
helloSensor/ index.html helloSensor.png
voiceExample/ index.html voiceExample.png
panorama/ index.html panorama.png

Figure 2. Argos applications contained in directories on the phone's memory card.

operating system capabilities, whereas WebOS is designed for HP's specific mobile platforms.

PhoneGap

PhoneGap is a Web-centric mobile development platform that uses HTML, CSS, and JavaScript. It supports various phone capabilities on multiple mobile operating systems such as iOS, Android, WebOS, Windows Phone, and Symbian. This reach is both a feature and a limitation. Because not all operating systems support the same features, PhoneGap chooses to support different sets of features on different platforms. Although Argos shares many of the same design principles as PhoneGap, it focuses purely on Android, providing comprehensive access to specific and powerful capabilities at the browser level.

W3C and HTML5

Advancements to the W3C's HTML5 standard, such as video, audio, device orientation, and geolocation, represent the ultimate push from native to Web-centric development. Web-centric platforms such as Argos, PhoneGap, and WebOS lead the way to these advancements by testing and maturing new capabilities. For example, these platforms could create APIs for nonstandard components such as NFC or camera-based head tracking, which could then form the basis for HTML support in the future.

ARGOS

The Argos system consists of a custom WebKit browser on top of an Android-specific JavaScript library that gives Web application developers access to low-level mobile phone capabilities. To accomplish this, Argos provides a Java-to-JavaScript bridge and a set of Java-based Android-specific managers to access native phone capabilities. Because Argos is Web-centric, people can develop applications using their desktop and an ordinary HTML5 browser. This gives them the flexibility, tool-chain, and development environment that they're used to.

Argos Architecture

In Argos, developers can create HTML, CSS, and JavaScript applications that are stored directly on the phone's memory card, so Argos doesn't require Internet connectivity to run. The Argos JavaScript library exposes the rich, but often complex, capabilities of the Android operating system in a clear and concise manner. For example, accessing the text-to-speech capabilities of the phone is as simple as writing:

```
Argos.getTextToSpeechMgr().say('hello, world');
```

Argos's primary rendering surface and rendering engine is Android's built-in WebKit, called WebView, with some modifications. When Argos starts, it looks for an `argos/index.html` file as the first page to show. The default uses a traditional grid of application icons, but by changing this page, you can create different ways of accessing and presenting applications. To create an Argos application, you simply create webpages and link the launch page to the Argos home page (see Figure 2).

Argos provides a set of JavaScript manager libraries that abstract access to the Argos Java-JavaScript bridge. For example, `Argos.NfcManager` provides functions to read an NFC tag's data and to write new data to the tag, and an event to alert your application when an NFC tag has been read.

Argos's Java layer is built around a set of Java-based managers to represent major components of Android. Each Java manager maintains its resources (for example, the camera resource). Android exposes a complex application life cycle through `onCreate()`, `onStart()`, `onPause()`, `onResume()`, `onStop()`, `onRestart()`, and `onDestroy()` methods, which handle interrupts from other applications such as a phone call. Argos instead exposes a simplified life cycle through `initialize()`, `turnOn()`, `turnOff()`, and `terminate()` calls, and implicitly handles the system state and resources. This is a cleaner abstraction, and it lets the developer explicitly turn resources on and off (for example, to preserve battery power or dynamically change security access).

To communicate between the Java and JavaScript layers, Argos uses a bridge construct that exploits Android's ability to expose Java methods as JavaScript functions. Each Argos Java manager passes values through this bridge using JavaScript Object Notation (JSON). This provides a uniform and generic interface between the back and front ends.

Overlays

Argos also supports three distinct rendering layers that are overlaid to provide one composite view (see Figure 3). The top layer is the HTML rendering layer (WebKit) that supports 2D graphics; the middle layer is an abstraction for OpenGL that supports high-level 3D graphics operations; and the bottom layer is the real-time camera view of the world. By superimposing all three, developers can create augmented reality applications with HTML controls embedded over the camera's view.

Emulator

Because Argos applications are written in HTML, CSS, and JavaScript, Argos can run on your desktop in an ordinary Web browser (see Figure 4). Argos automatically detects whether the application is running on an Android mobile device or on a desktop. If it's running on a desktop, Argos is emulated by the browser as a webpage view that is constrained to be the size of the phone screen. The emulator also simulates any data that you would get from a sensor. This provides tremendous power for developers, as they can use their favorite Web IDEs, libraries (such as JQuery and dojo), and debugging tools to rapidly prototype and test an application. When they want to run the application on an actual phone, they can simply copy it from the desktop to the phone's memory card.

We've already produced several Argos applications with novel user experiences that combine various distinct phone capabilities. Even at this early stage, we've found it beneficial to rapidly prototype applications using a system that integrates fluidly with the Internet, allows access to low-level capabilities, and provides a consistent and clean abstraction.

There are several different areas in which Argos can expand. For developers, we see the need for a higher-level IDE that lets them drag-and-drop modules to build the basics of an application before coding in earnest. We also see an opportunity to explore and improve on existing mobile security models; for example, making access to resources and applications depend on the context of use.

For users, Argos could lead to greater flexibility and customizability of their phone's user interface. The fact that Argos, including the application launcher, is really just a set of HTML, CSS, and JavaScript files means that many more people have the skills to change their phone's look and interaction style. This could potentially lead to a robust themes marketplace, letting users deeply customize their phones by simply installing one of thousands of themes. This would be similar to WordPress's themes directory (http://wordpress.org/extend/themes) for customizing WordPress blogs.

Argos, PhoneGap, and other Web-centric platforms provide a way for Web developers to easily incorporate mobile functionality into their applications. The most popular functionality will most likely be incorporated into HTML standards, with these platforms leading the way in the evolution of mobile, pervasive computing.

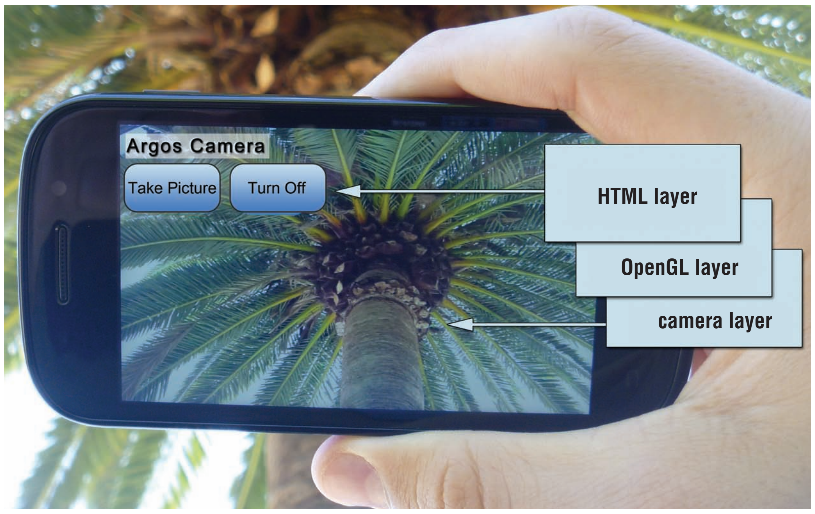


Figure 3. Argos provides easy access to multilayer rendering to support augmented reality and other related applications.

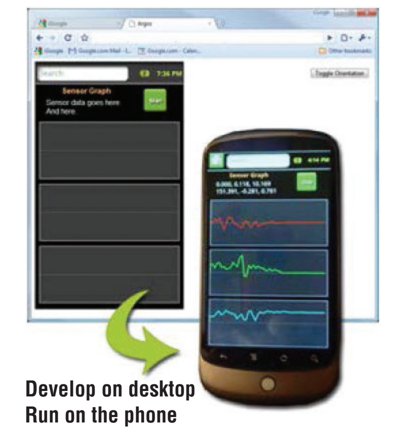


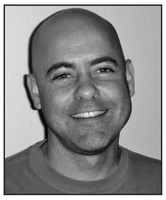
Figure 4. You can develop in Argos on the desktop and then copy to the phone's memory card.

REFERENCES

1. Hewlett-Packard, *An Overview of HP WebOS*, https://developer.palm.com/content/resources/develop/overview_of_webos/overview_of_webos.html.

2. World Wide Web Consortium, *HTML5 W3C Working Draft*, 25 May 2011; www.w3.org/TR/html5.

3. B.R. Schatz and J.B. Hardin, "NCSA Mosaic and the World Wide Web: Global Hypermedia Protocols for the Internet," *Science*, vol. 265, no. 5174, 12 Aug. 1994, pp. 895–901.



Rich Gossweiler is a research scientist at Google. His research interests include mobile and social computing. Gossweiler has a PhD in computer science from the University of Virginia. Contact him at rcg@google.com.

Colin McDonough is a software engineering intern at Google. His research interests include mobile and Web applications. McDonough will complete his BS in computer science at Washington University in St. Louis. Contact him atcmcdonough@wustl.edu.

James Lin is a software engineer at Google. His research interests include end-user programming and user interface design tools. Lin has a PhD in computer science from the University of California, Berkeley. Contact him at jameslin@google.com.

Roy Want is a research scientist at Google. His research interests include mobile and ubiquitous computing, wireless protocols, embedded systems, and automatic identification. Want has a PhD in computer science from Cambridge University. He is an IEEE Fellow and an ACM Fellow. Contact him at roywant@acm.org.