



The Challenge of Creating a Single Brand across Multiple Mobile Devices

SurfKitchen's CTO, Dave Evans Reviews the Challenge of Creating a Single Brand across Multiple Mobile Devices

One of the mobile industry's major goals is one of its biggest concerns right now: How to create a single mobile experience that works every time, on every platform and every device.

If history is any judge, the industry has a right to be concerned. PCs have had the user-experience problem since the dawn of the personal computer. It continued (many think got worse) when the Internet browser was introduced. This problem is magnified on mobile devices where there is a much greater variety of both platforms and browsers.

Mobile application developers typically use the device's core controls—buttons, list boxes, grids to create their application resulting in applications that 'look' and 'behave' differently on different devices. Even what you read can be a challenge: Some devices support true type scalable fonts and some only bitmapped fonts.

This platform fragmentation does little to boost consumer confidence in the industry and doesn't help operators and media publishers, who want to create a single experience which they can promote broadly and boldly to reinforce and build brand loyalty. The launch of Microsoft Windows clearly demonstrated the value of a universal platform in the PC world, as despite detractors pointing to the absence of competition, it provided a single user experience that drove the growth of applications on personal computers.

The twin challenges: Coherent Services and Consistent Communications

What makes the mobile market so complicated and fascinating is the challenge of creating coherent and compelling cross-platform services in the face of fast changing technology.

Service developers have always had to choose between targeting the broadest possible audience by limiting their service to the lowest common denominator, or creating a great experience on just a few platforms. For broadest reach, service providers have built their service within a browser, allowing them to run on a broad range of devices. However a browser has no 'knowledge' of the device it is running on. It doesn't know, for example, whether the device has a camera, GPS, Bluetooth, WiFi, storage and the like, and so the service cannot leverage these capabilities. The alternative has been to build the service in the 'native' environment on the device—Symbian, J2ME or Microsoft. Such services can leverage the device capabilities but rarely can be used on other device platforms or even devices.

Then there are the technology challenges. Mobile networks' characteristics are different than wired networks. The connection between a client and server has greater latency and connections can fail due to interference, battery failure or congestion. This is particularly noticeable in HTTP connections as HTTP is quite 'chatty.' Each asset in a Web page requires a new HTTP request, typically with a DNS [Domain Name Service] lookup, loading the mobile network with session setup and communication. A typical Web page will require 10-20 individual assets—images and text requiring 10-20 HTTP requests with all their overhead to be initiated. Mobile application developers need to handle multiple failure conditions for each of those requests, which often forces them to write complex and unwieldy code.

SurfKitchen tackles these challenges head on by taking the common device capabilities and 'wrapping' these capabilities within a device object model accessible from a Javascript engine. This enables the service developer to build their service in Javascript/XML in a similar manner to a browser service, but access and use the device capabilities if they are available. This delivers the best of both worlds—a high-level productive language with which to build services, a broad range of devices that can be targeted, as well as deep device integration allowing compelling services to be rapidly built and deployed.

The core SurfKitchen runtime engine contains a number of common controls that are consistent across platforms and font-display capabilities that enable a brand owner to create a consistent experience across Brew, J2ME, Symbian, Blackberry, Microsoft and in the future Android. With these common controls, a service developer can craft a solid foundation on which to build its experience—consistent font technology, consistent behaviour even across keypad devices and touch screen devices.

To address network communications, SurfKitchen developed a multi-file protocol running over HTTP. The client makes a single request to the server, the server packages up all the files required for that request – images, XML, Javascript etc., into a single HTTP response stream, minimising the 'chatty' nature of HTTP. Optionally the SurfKitchen Server can return a complete set of files, or just the delta required by the client further optimising the data transfer. The client receives this stream of data and reconstructs the individual files with full error detection. In the event of partial transmission the client can easily request the data again, with full resumption – ensuring only those files not received are re-sent.

By providing an environment that delivers a common set of controls while optimizing communication, SurfKitchen allows the service developer to focus on building a compelling service, rather than the 'plumbing'.

Compilation versus interpretation

There are two methods of executing software. The first is a compiled programme, where the code written by a developer is 'compiled' into bytecode which the device understands. This bytecode or executable is delivered to the device and executed. Compiled code typically runs fast as the device 'understands' the code it is written in. The second option is interpreted. With interpreted code, the developer writes code in a high level human readable form. This code is then delivered to the device, and the device interprets the code into

bytecode at run time, rather than before. Typically this is a slower process than compiled code.

So why use interpreted code? Interpreted code has a number of advantages over compiled code. Firstly it can be platform independent – as the platform itself will interpret the code when it runs. Compiled code needs to be pre-built for a target platform. Secondly as it is interpreted at runtime, the program can easily modify itself and adapt to different screen sizes etc. Thirdly and very importantly for mobile data applications, part of the application can be delivered separately, or dynamically from a standard Web server, or actually created on the fly. An example of the difference is a Web application which is made up of a series of XHTML pages. The browser interprets these 'on the fly' to render the experience to the user. At any time the developer can change one of these pages, and the experience for the user will immediately change. Compare this with a Flash Lite experience where a developer builds a Flash 'movie'. This movie is compiled into a FLA Flash file and delivered to the device. If the developer wishes to change a part of the experience, they must go through this complete build/compile/deliver cycle again.

SurfKitchen provides an XML/Javascript interpreter on the device. This enables the client to retrieve XML and Javascript from any standard Web server and incorporate this into the overall experience for the end user. SurfKitchen is one of very few vendors providing a full interpreted Javascript experience across J2ME, Brew, Symbian, Microsoft and Blackberry.

Conclusion

The most urgent matter before the mobile industry today is the user experience. Many companies are trying to tackle it from one angle or another, but SurfKitchen, we believe, has addressed the twin roadblocks to a single user experience, coherent services and consistent communication protocols.

By addressing the challenges at the kernel level and through HTTP, SurfKitchen has built a foundation on which companies can develop a complete on-device branded experience that will enable subscribers to easily discover, preview and purchase content and applications. That foundation provides an intuitive user experience and gives operators and developers the ability to simply conceive, build and deploy mini-applications.

This is exactly the kind of openness and standards-based flexibility that the industry has been looking for to help it thrive in the coming years.