HTML5 Framework and Gateway Caching Scheme for Cloud Based UIs

Mike McMahon, VP of Web Experience and Application Strategy
Charter Communications

Abstract

Recent advances in our industry such as TV Everywhere and second screen, companion apps merge video delivery and consumption with web technologies. Similarly, much progress has been made in introducing service-oriented architectures, exposing common web services and enabling a high degree of consistency and re-usability in backend systems.

Video is now clearly being consumed on a wide range of devices and these devices can vary wildly in terms of screen size, capabilities, development platforms, etc. Tablets, game consoles, smart TVs, mobile phones and a number of other devices are all viable video terminals. Processing and delivering video into a variety of flavors, bit rates and such is non-trivial but is generally well understood and now fairly commonplace. In order for the Cable industry to fully embrace an already highly fragmented client platform landscape and position itself to exploit new devices as they become available it is necessary to achieve a similar level of abstraction and re-use in the way user interfaces are built, delivered and maintained. This paper presents an HTML5 based UI framework, built on open standards but optimized and configured specifically for the needs of TV centric applications.

THE HTML5 OPPORTUNITY

Although HTML5 remains a maturing technology, the web development community has actively embraced it and most modern web browsers already support it. In our industry, there has been some speculation surrounding the video tag and the current lack of DRM. The premise here is not "HTML5 video," rather the use of cutting edge web techniques associated with establishing a user interface, built from a singular and re-usable code base which is common and consistent across a wide range of devices. For the Cable Industry, moving the user interface code into the Cloud in this way not only represents an opportunity to address a variety of devices, but additionally empowers us to embrace retail devices as well as add features and extend functionality at web-like velocity, removing the burden of code downloads and complex provisioning scenarios.

Open Source Frameworks

There are countless examples of extremely powerful applications, written entirely as web applications that are as rich in functionality, animation effects and behavior as desktop applications. In practice, these are written as a combination of HTML5 along with an aggressive use of JavaScript and CSS3. It is important to recognize that it is this collection of technologies, rather than HTML5 itself that enable these types of user interfaces. A variety of JavaScript and CSS3 frameworks such as jQuery or Sencha exist for HTML5 development. These typically abstract away platform idiosyncrasies, establish object and state models, provide a variety of animation libraries and generally simplify the development of a single web application to run across a variety of devices.

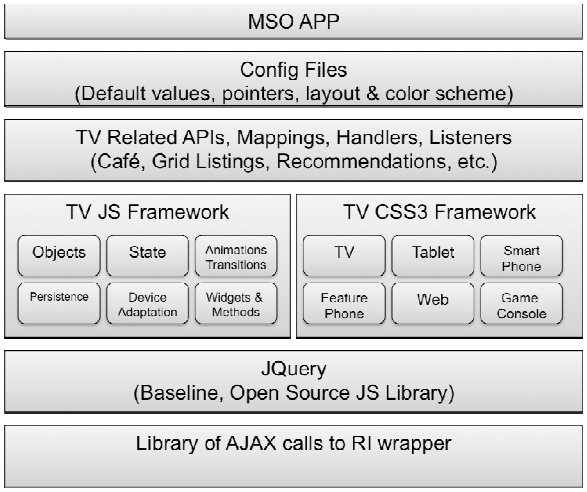
THE GAP

Without doubt, individual providers will seek to differentiate their brand through unique designs, features and interactivity. While each individual service provider could certainly select a given framework and develop its own, unique cross platform user

interface there would be little commonality across the industry and much duplication of effort. We will all have linear listings and VOD search. We will all have cover art and DVR scheduling. Grids and a baseline set of animations are inevitable. We will share the need to support the same range of devices. Furthermore, each provider would be burdened with updating to versions of the framework and addressing new devices, screen resolutions, etc.

AN INDUSTRY FRAMEWORK

Envisioned here is a Cable Industry UI Framework. The ambition is to select among the various open source HTML5 frameworks the core aspects most beneficial to the generic needs of TV centric user interfaces. There would likely be several components involved, the particular assembly of which would constitute an MVC type construct with particular focus on the device and object abstractions required to represent "TV." This baseline component assembly would constitute the core foundation but would require an additional layer of CSS3 and JavaScript abstraction for the Cable specific UI components and underlying object model. The high level stack is represented below:



The overriding purpose of this stack is to leverage the generalized foundations of an underlying open source framework such as jQuery and build on top of it the necessary specifics relating the Cable industry. These specifics would include such things as objects for TV listings, recommendations, actors, movies as well as standardized methods and callback routines for fetching recommendations, content searches, etc. Likewise, a variety of UI components representing things like an actual TV listings grid and animation effects such as a cover art carousel would be optimized. CSS3 style sheets for each device family or particular model would cater to the specifics needed in each rendered component. Each layer of this stack is intended to be extensible.

MSO Customization and Extensibility

Each MSO would benefit for the shared plumbing in the underlying framework. Configuration files, unique to each MSO would map to their web service endpoints, define the specific assembly of the various UI components into their presentation and provide a skinning capability via CSS3 overrides.

As new objects, event handlers or animations were envisioned and required; an MSO or third party would develop them within the overall framework. Ideally, these would be contributed back into the community such that other MSOs would benefit. There would likely, for example, be several variations of a TV listings component to choose from as well as useful extensions by way of animation effects.

Inclusion in App Stores

There is often confusion between "Apps" and "HTML5." The two are indeed different things as "Apps" are compiled, installable binaries and "HTML5" represents web pages. App stores and HTML5 are, however,

perfectly compatible. There is, of course, very good reason to place applications in app stores. Most users of iOS and Android devices in particular are now familiar with app stores and this is the dominant avenue by which they are likely to search for and discover an MSO application. Applications available in these stores are compiled natively for the specific platform. In order to achieve the benefits of app store inclusion as well as the ability to re-purpose HTML5 across platforms a "native wrapper application" is written which essentially compiles a rudimentary shell for the specific platform and uses the device's underlying web browser to render all actual user interface screens. This is, for example, the way in which Netflix develops its applications.

ADDRESSING THE BIG SCREEN

With regards to the common retail devices of today such as iOS and Android powered smartphones and tablets, laptops and PCs this framework would provide a robust mechanism to deliver a common and consistent user interface as well as minimize the associated code. The UI is, effectively, a giant web site delivered from the Cloud. Changes made to a single file would propagate to all devices and users would not need to download any updates, they would simply benefit from the new experience during their next session. This is all well and good, but to what extent could the framework be used to deliver the same experience to a STB connected to a 60-inch plasma?

Relevance to the RDK

The RDK recently introduced by Comcast includes a Webkit implementation. Webkit is an open source HTML5 compliant web browser, used in both Apple's Safari and Google's Chrome browsers. This provides an alternative to Java as the presentation environment on the CableLabs <tru2way> reference implementation.

This stack can be used in a master-slave in-home architecture whereby a gateway device running the full RDK stack serves as the service termination point within the home, commanding control of tuners, handling conditional access, etc. Additional devices such as laptops, tablets and smartphones can connect to the gateway and both consume tuners and receive the user interface, which is delivered as HTML5 via a web server running inside the gateway. The gateway can be "headed" meaning a television display is actually connected to it or "headless" meaning it serves as the termination point but exclusively provides the UI and services to other devices within the home. Additional outlets need only be very dumb, thin IP STBs, which run Webkit.

To be sure, this description of the RDK does not do it full justice as it is, truly, a very compelling development and much more significant than the brief description above. The point here, however, is that there is an HTML5 presentation layer available and that it can render to the big screen.

Thus, a modern HTML5 compliant web browser is available through the RDK. The RDK, however, does not provide any specific UI or further framework, just that open book upon which things could be written. As is the case with other HTML5 devices, each MSO would need to develop and maintain its own specific UI.

The proposal is to extend the RDK to include the same UI framework discussed in this paper.

GATEWAY CACHING SCHEME

The UI framework would necessarily be hosted in the Cloud. This would allow it to be used independent of RDK gateway architectures as well as ensure that the UI

could be rendered outside of the home over any network. By including the framework within the RDK, however, there are additional benefits relating to caching and performance to explore.

Insofar as an RDK based gateway acts as a web server (both to itself and other devices within the home) it is, in effect, a proxy to the actual remotely hosted Cloud. Like all proxies, it acts as the source of truth from the perspective of the client. This presents potential challenges by way of ensuring the gateway is, in fact, up to date but also represents a significant opportunity to be used as a caching node within a distributed architecture. Open source caches such as Varnish could be additionally included within the RDK and would provide a tremendous performance benefit. Specifically, the gateway could be configured to proactively cache guide and VOD listings, cover art, network images, user profiles and targeted ads. This could be done relatively easily via a lazy cache whereby the gateway deferred to the Cloud and simply stored content and data as it passed it through to the client, making it locally available for subsequent requests. It could also take on a more elaborate form whereby server side algorithms proactively pushed information to the gateway, likely during dark hours and with certain targeting parameters designed towards personalization of the UI.

ADDITIONAL CONSIDERATIONS

While I believe the industry would benefit significantly from a common, shared core UI framework it still assumes HTML5 and relatively modern web browsers. What about older PCs or even fairly modern devices with limited rendering capabilities like Smart TVs or game consoles?

HTML5 does not render everywhere. Older browsers like those in many PCs or early incarnations of Smart TVs are capable of rendering simpler versions of HTML. It is necessary that the framework can degrade gracefully by recognizing these devices and rendering a simpler, less animated form of the UI. This would require a somewhat more complex abstraction than would otherwise be necessary but is perfectly feasible. Other devices, like a current Xbox, will require platform specific applications. These devices will benefit from at least a general commonality of the UI in terms of data and objects as served from backend web services, but would still require platform specific, native applications to be written. HTML5 will not currently provide a UI on every device; although it will address a wide range of current devices and it is likely Webkit will continue to proliferate to things like Smart TVs and game consoles.

The Vulgarity

More challenging to the notion of a common UI framework is the fact that there is currently very little consistency in the backend of MSOs. While most of us are embracing web services and these web services are notionally representing very similar things they are far from standardized. The grid for example, is assumed to be in most operators' UI in some form or another. The data used to populate the grid would be fetched through a web service along the lines of:

```
http://operator.com/apis/getGrid();
```

The host and specific method call, of course, would be perfectly configurable and each MSO would have their own unique endpoint. This is not a problem. The syntax and structure of the response, however, is a challenge. There are differences in semantic naming conventions as well as overall object models. The semantics of one MSO labeling HBO a "network" and another "provider" or "programmer" are somewhat easier to deal

with. Structural differences in the objects or varied sets of interfaces are far more troublesome. One MSO might include actors and detailed descriptions in the getGrid() response. Another might have a secondary web service for getAssetDetails() and a third that does not include actors at all.

These backend variations are not insurmountable but they do require some additional thought. Standardization of core web services is, of course, the ideal solution. It is also possible to establish a JavaScript mapping layer within the framework, although that will likely lead to poor performance. A possible middle ground scenario would involve each MSO establishing a server side transformation layer to its existing web services.

CONCLUSION

As MSOs, we face a similar challenge in providing consistent user interfaces to a growing set of devices. Cloud based UIs allow us to more uniformly deliver and present a user interface as well as extend new features and services in a coherent and efficient manner unlike with traditional STBs. This is something we can immediately explore online and through smartphones and tablet devices and will increasingly become viable on leased CPE through initiatives like the RDK. A common, shared industry UI framework would allow us to further exploit the opportunity and reduce the individual burden of redundant web development. Such a framework could be developed based on best in breed, open source efforts from the web community but configured specifically to suit the needs of TV centric user interfaces.