HTML5 and Device APIs for Automotive: Is it time to power Infotainment and Car Portal Applications with Web Technologies?

Diana Cheng – diana.cheng@vodafone.com

Introduction

A key advantage of HTML5 and Device APIs is their potential to produce rich cross-platform applications with a single core codebase. Although customization will be needed to make web applications run similarly enough on different devices, the development process is simplified in that only one project is managed, only one skill set is required (HTML, JavaScript and CSS) and developers with web expertise are one of the biggest groups available. Web applications are also attractive in that it is easy and immediate to push updates to them, be it new features or bug fixes. All that it takes is to update the application in the hosting server and the changes are immediately available to all users without any need for over-the-air upgrades.

We are starting to see the emergence of complete standalone full-web OSes for both desktop (e.g. Chrome OS) and mobile (WebOS, Firefox OS [1], Tizen [2], etc.). These platforms are pushing the existing standards implementations as well as implementing deeper integration with the device by eliminating the overhead of the native layer and thus potentially making web applications highly performant without the need for high-end devices. Automotive is a space that has seen a very fragmented technology ecosystem; there are currently a number of line-fit systems based on proprietary platforms as well as others based on Android, Windows, QNX, etc. These platforms could be enriched and extended with new applications and services but they often currently restrict any extension capabilities. This makes automotive a niche where opportunities are still available to create a convergent platform where all parties can benefit from a standardized and open approach to application development.

Automotive services go beyond devices and applications that are used inside the car. Each of the devices that a user owns, e.g. mobile phones, tablets, TVs and PCs can enhance the experience and each of them provide a particular ease of use for a specific piece of functionality of the entire automotive experience. For example, the user might be more comfortable searching for directions on his PC at home before a trip and sending them to his in-car device so they are automatically loaded before he starts his trip. Conversely, the user can continue with his in-car session (music he was listening to, walking directions) on his mobile phone upon leaving the vehicle.

Given this diversity of devices, which might encompass a number of different platforms, web applications present a unique opportunity to accelerate advancement in this sector and bring automotive applications and platforms to the masses; web applications can, for example, be adapted in their user interfaces to the device using a number of existing techniques in order to present the best experience for every form factor.
Moreover, there seems to be a preference amongst makers of IVI (in-vehicle infotainment) systems to put non-critical applications at a higher level in the application stack, in a secure sandbox [3] [4] (e.g. running within the browser's sandbox) in order to restrict them of access to the rest of the system so they cannot interfere with critical car applications that run directly on top of the native OS.

As of today, it is often believed that web technologies including HTML5 and browser implementations of Device APIs cannot provide the same level of user experience and performance that native applications do, but that is highly dependant on the application. Therefore, in this paper, we present a few use cases for In-vehicle Infotainment and Car Portal applications. We comment on the state of web technologies, to which extent they can help fulfil these use cases, and for which of them more development and work is needed to provide a great user experience.

In-vehicle Infotainment Applications

This section refers to applications used while the car is being driven, whether used by the driver himself or other passengers. IVI systems typically include components such as turn-by-turn navigation, (Internet) radio, Bluetooth synchronization for phones and music devices, social media integration, navigation via voice commands, etc.

Applications for IVI systems are nonetheless not restricted to the driver. Targeting passengers presents other opportunities for monetization such as purchase of paid content like games, movies, music, etc.

1. Session handover: an important aspect of the user experience in automotive applications is for the user to be able to continue his in-car session when leaving the vehicle, e.g., the user might want to continue listening to his music or radio on his mobile device or he might need walking directions to his final destination after having parked the car at an available location which might be far from the site he really wants to reach.

This automatic session-handover can be realised both with native and mobile web technologies. On Android for example, a push notification server can send a "wake-up" message to the mobile device which will start the music or map application for it to pull the necessary information to carry on with the session, such as minute and second of the track being listened to, or the car's actual and final destination locations in order to plot and present to the user the relevant walking directions.

Server-Sent Events [6] provide a mechanism for the browser to receive standard DOM events from the server even when the application is in the background so they provide a way to similarly synchronize web applications for the session handover described above.

WebSockets [6] provide a full duplex communication channel between a web server and the browser, and this can be used as well when a bidirectional communication is required, such as, besides from receiving events from the server, also pushing the device location, for example. The downside of WebSockets is that the "web page" needs to be active in order for the bidirectional channel to remain open, i.e. it needs to be in the foreground and if it's run within a multi-tab browser, the tab needs to be active, otherwise when the timeout is reached the connection will be lost and no further updates will be delivered. In a session-handover scenario this means that upon leaving his vehicle, the user needs to actively start the music or map web application and only then the session synchronization will take place via WebSockets.

Although libraries and polifylls exist, both Server-Sent Events and WebSockets still lack support amongst mobile browsers [6].

2. Speech recognition: Voice is the main interface for a driver to interact with an in-vehicle infotainment system, as it provides minimal distraction. Off-the-shelf products with different levels of accuracy are widely available. These are typically embedded applications running on selected platforms and hardware [8].
For Web applications to make use of speech recognition the W3C Speech Input API spec [9] was developed. Even though the spec is agnostic of the underlying speech recognition implementation (server-based or embedded) currently the only available implementation is in the Chrome browser which accesses a Google backend server, i.e. the captured audio is uploaded and a usable transcribed string is returned.
This server-based mechanism is applicable for in-vehicle applications as long as Internet connectivity (and a reasonable bandwidth) can be guaranteed. Given that connectivity in the car is provided by cellular networks of variable coverage according to the location (possible bad signal in the countryside, going through tunnels, etc.) and the high speed at which cars move, relying 100% on backend Internet services becomes unpractical, especially as a speech interface should be available at all times. At the minimum, basic speech recognition capabilities should be available offline, while the majority and the most advanced features could still rely on a backend server. We are starting to the emergence of such offline support in mobile OSes, such as in the latest version of Android, 4.1 or Jelly Bean [10].

Text-to-speech (TTS) is also a requirement for tasks such as reading of incoming SMS, providing real-time updates and for giving driving directions (as discussed in the next item), and it encounters the same limitations described above for speech recognition. For example, the Google Translate API provides TTS capabilities and it can be used to send the text via HTTP and receive back an mp3 file to be played in the browser, both of which require Internet connectivity in order to work. There is yet no currently implementable spec in the W3C for TTS; Google previously proposed one (see: [11]), but after the W3C HTML Speech Incubator Group issued its final report [12], there is a new proposal which encompasses both speech recognition and synthesis for web applications [13] and the possibility of a Community Group being formed. So we might see this becoming standardized and supported in browsers in the future.

3. Navigation Systems: Even though sophisticated dedicated navigation systems

are becoming accessible to most drivers in mature markets, HTML5 provides today sufficient capabilities for several of the features of a purely-web navigation system.
The W3C Geolocation API [14], widely implemented in most desktop and mobile browsers, provides access to the device coordinates, and allows keeping continuous track of its location.
Web workers [15] could be used to calculate routes and to keep track in the background of whether a recalculation is needed.
AppCache (Offline Web Applications) [16] of the HTML5 spec and Local Storage [17] could allow car applications to locally store map assets, page assets and navigation and application data to reduce as much as possible unnecessary traffic, page loading time or to offer functionality when no connection is available. That said, in terms of performance, it's yet to be seen if the browsers' offline capabilities can cope with and provide good performance when dealing with advanced web applications such as a web navigation system. There have been a number of reported performance issues with Local Storage [18] (particularly when storing large amounts of data) including its synchronous I/O nature (which can block websites) and the limited default storage space in the available implementations (which is not discoverable in a programmatic way). Moreover, Local Storage is aimed at saving text data. This can prove beneficial for text assets such as JavaScript and CSS files, but storing encoded binary assets is discouraged. Although alternatives like IndexedDB [19] and WebSQL [20] exist and do add support for binary data, their implementation in desktop and mobile browsers is not yet universal and they might not adapt to the use case in terms of user experience (IndexedDB requires prompting the user for permission and that might be undesirable for some applications). Moreover, IndexedDB is hard for developers to use and it's not implemented in most mobile browsers, which in turn implement the now deprecated WebSQL standard.

4. Installable applications: Another desired feature of an in-vehicle infotainment system is extensibility: the ability of the driver and passengers to add functionality by downloading and installing apps in order to access them from the home screen of their in-car device.
Applications built using standard HTML(5), JavaScript, CSS, etc. can be packaged using open standards like the W3C Widgets specification [22], can be installed on the user's device and will run using the browser's rendering engine, but without displaying the browser's chrome. However, widgets as self-contained packages do not benefit from the immediate distribution and update facilities of hosted web apps which we have mentioned at the beginning of the paper.

Another way to build installable applications with standard web technologies is to use an open-source framework such as PhoneGap [23] now moved to the Apache foundation under the name of Cordova. PhoneGap acts as a native wrapper for the web application thus making it a 'hybrid' one. Since applications are wrapped as native they become discoverable via native mobile application stores and installable in a native mobile platform such as Android. Like this, an automotive platform built on top of Android could be extended using hybrid apps.

Car Portals

This section refers to the applications used outside the car that allow the user to visualize aggregated information about his vehicle usage. This typically includes: trip logs, current car location, diagnostics information obtained via OBD2 devices such as gas and oil levels, mileage, car error messages, etc.

1. Map Visualization and touch interaction in mobile web applications: Visualization of completed trips, location of the vehicle or visual real-time vehicle tracking, all of them on a map view are some of the basic use cases of a car portal both on desktop and mobile. As mentioned, browsers already provide the capabilities for rich UIs that can include map visualizations (this can be achieved by use of several existing JavaScript APIs such as Google Maps, OpenLayers, etc.). A car portal application requires, therefore, that for mobile browsers, support for interacting with maps via touch is be available. As off today, some "modern" mobile browsers such as the Android Browser in Gingerbread (Android 2.3) do not support multitouch events. This results in very poor UX when interacting with maps for the use cases mentioned above: it is not possible to pinch-to-zoom, and double tap feels slow and awkward. Without support for touch events and other standards in mobile browsers, there is little point in working on and refining web specifications like it's been done with the W3C Touch Events Specification [24].

2. Real-time Status Notifications to mobile devices: Automotive applications may want to notify the user directly of

specific events related to his vehicle, for example, when a second driver starts the car, or to send notifications to an emergency contact in case of a potential crash detected.
Automotive systems should push this kind of notifications to mobile devices running a Car Portal application in a way that captures the user’s attention (for emergency warnings for example), preferably displayed as status bar notifications in his mobile device. This kind of notifications is at the moment only available in native mobile stacks. There is currently no way for web servers to send updates to a mobile web application and alert the user outside of the “web page” or browser context, such as notifications displayed in the status bar of the device or on its home screen. A W3C specification that is being worked on in order to enable this, is Web Notifications [25]. The spec has recently received attention from relevant implementers such as Apple but it is currently only in Working Draft status so we might need to wait a while before we start seeing interoperable implementations amongst mobile browsers. Chrome browser for Desktop is currently the only implementation of the spec available.
Also, a device API which would be useful when sending real-time notifications to users of mobile devices is the Vibration API (currently being worked on at the W3C DAP Working Group and in First Public Working Draft status).

Conclusions

As detailed above, web technologies such as Geolocation, AppCache, Local Storage, Server-Sent Events and WebSockets help us build rich web applications with open standards. In spite of that, several of the use cases for automotive described here are currently still not covered.

From our **point of view**, the following technologies require more work in order to fulfil the presented requirements:

- An agreement as to which spec and mechanism should be used for storing binary data in web applications (either IndexedDB or WebSQL). Implementers should then support the same standard across desktop and mobile browsers.
- Advance and finalize the Web Notifications spec and support it on mobile browsers (currently there is no implementation [21]) which coupled with Server-Sent Events or WebSockets would provide analogous functionality to Native Push notifications that can be displayed on the status bar of a device.
- Broader support for Web Sockets and Server-Sent events amongst mobile browsers.
- A mechanism for automatic update of Widgets so they can benefit from the immediate deployment that hosted web apps enjoy.
- Agreement on and standardization of a spec for in-browser text-to-speech.
- We require browser implementations to support advanced touch interactions, specifically the full W3C Touch Events spec.
- Advanced support for offline speech recognition, and not just provided by backend services as in current implementations. This remains a strong requirement throughout a number use cases in IVI platforms, and it is here where web technologies don’t fully meet the needs yet.

Even when a full web automotive platform is not feasible today, we believe that improving the items mentioned above would make the web a stronger contender for an automotive platform and related applications.

References

[1] Firefox OS/Boot2Gecko: https://wiki.mozilla.org/B2G
[2] Tizen Platform: https://www.tizen.org/about
[3] Sandboxing for Automotive: http://lwn.net/Articles/465316/
[4] In-vehicle infotainment software architecture: http://goo.gl/Qwsau
[5] Server-Sent Events: http://dev.w3.org/html5/eventsource/
[6] Web Sockets: http://www.w3.org/TR/websockets/
[7] Server-Sent Events support: http://caniuse.com/#feat=eventsource
[8] Nuance Homepage: http://www.nuance.com/for-business/by-product/automotive-products-services/nvc-auto/index.htm
[9] Speech Input API: http://lists.w3.org/Archives/Public/public-xg-htmlspeech/2011Feb/att-0020/api-draft.html
[10] Offline Speech in Android Jelly Bean: http://www.readwriteweb.com/archives/google-i-o-android-41-aka-jelly-bean-will-have-responsive-widgets-offline-voice-typing.php
[11] Text to Speech API proposal: http://lists.w3.org/Archives/Public/public-xg-htmlspeech/2011Feb/att-0022/htmltts-draft.html
[12] W3C HTML Speech Incubator Group Final Report: http://www.w3.org/2005/Incubator/htmlspeech/XGR-htmlspeech-20111206/

[13] Speech JavaScript API proposal: http://lists.w3.org/Archives/Public/public-webapps/2011OctDec/att-1696/speechapi.html
[14] W3C Geolocation API: http://dev.w3.org/geo/api/spec-source.html
[15] W3C Web Workers: http://dev.w3.org/html5/workers/
[16] Offline Web Applications: http://www.w3.org/TR/html5/offline.html
[17] Web Storage: http://dev.w3.org/html5/webstorage/
[18] Performance issues of Local Storage: http://hacks.mozilla.org/2012/03/there-is-no-simple-solution-for-local-storage/
[19] Indexed Database API: http://www.w3.org/TR/IndexedDB/
[20] Web SQL Database API: http://www.w3.org/TR/webdatabase/
[21] Suport for Web Notifications: http://caniuse.com/#feat=notifications
[22] W3C Widgets: http://www.w3.org/TR/widgets/
[23] PhoneGap: http://phonegap.com/
[24] Touch Events Specification: http://www.w3.org/TR/touch-events/
[25] Web Notifications: http://dev.w3.org/2006/webapi/WebNotifications/publish/WebNotifications.html
[26] What is a Polyfill: http://remysharp.com/2010/10/08/what-is-a-polyfill/