The Architecture of a Web-Based Ecosystem and Its Challenges

Dr. Hermann Simon Lichte – hermann.lichte@net-m.de
Kornelius Dridger – kornelius.dridger@net-m.de
Patrik Schulze – patrik.schulze@net-m.de

Invited talk at the Mobile Forum Stuttgart (MFS)
Haus der Wirtschaft – May 17th, 2013

Motivation

→ Smartphone sales have outperformed PC sales, hence the web tends to be used more and more from smartphones.
→ The dominant mobile platforms today are not truly open and controlled by big Over-The-Top players (OTTs).
→ Apps on these platforms “do not speak” the language of the web: HTML5, CSS, and JavaScript are available in any browser.
→ End-users, developers, and Mobile Network Operators (MNOs) are forced to play by the rules set by the OTTs – with no exception.

The Vision:

→ Establish a completely open mobile environment where everyone can participate as they wish, without having to pay for using the technology.
→ Web techniques lower the entry barrier of app development.
→ Web developers can distribute their web apps on their own terms and conditions.
→ End-users have more freedom of choice.
Towards a Truly Open Mobile Ecosystem

“With HTML5, the open web platform is getting more and more powerful. All applications should be open and web-based, not closed in native OS environments.”

– Sir Tim Berners-Lee, founder of the Web and overall director of the W3C

Berlin, 25.08.2012

“The Web is the platform.” – Mozilla

Outline

The architecture of Firefox OS

Mobile Ecosystem (Firefox Marketplace & Co.)

Web-Based Architecture (Gaia / Web APIs)

Low-Level Architecture (Gonk / Gecko)

Focus of this talk

...and its challenges
Market Situation in 2013

→ European market:
  → Deutsche Telekom starts selling the Alcatel One
    Touch Fire running Firefox OS in Poland this summer.
  → Telefonica will launch Firefox OS in Spain.
  → Telenor focuses on Hungary, Serbia, and Montenegro,
    stimulating the growth of local development communities.
  → More countries in eastern Europe shall follow suit.

→ Latin American market:
  → Together, Telefonica and ZTE will bring Firefox OS devices
    to Mexico, Venezuela, Colombia, and Brazil.
  → Why these countries? They have been dominated by feature
    phones and smartphones are just emerging.

In these developing markets, the cost of Firefox OS smartphones shall be similar to
the cost of feature phones. Thus, Firefox OS should become the entry-level
smartphone.

Firefox OS High-Level Architecture

Gaia
- Status Bar
- Notifications
- Lock Screen
- Window Manager
- Trusted UI Manager
- Keyboard Manager

Gecko
- Chrome Process
- Document Object Model (DOM)
- Layout Engine
- Gecko Hardware Abstraction Layer
- Cross-Platform Component Object Model (XPCOM)
- Media Decoder

Gonk
- Gonk Hardware Abstraction Layer
- OEM Drivers
- Radio Interface Layer (RIL)
- Linux Kernel
- Baseband Firmware
Firefox OS Radio Interface Layer

Firefox OS uses the Android Radio Interface Layer for accessing the radio hardware.

- **RIL Daemon** talks to vendor-specific RIL library and dispatches messages from / to HW:
  - **Solicited requests** such as dial and hangup
  - **Unsolicited responses** such as new SMS
  - Access via RIL Proxy using Unix domain socket (/dev/socket/rilproxy)

Gaia
- System
- Dialer
- SMS

Gecko
- Messages
- Telephony
- SMS
- Radio Interface Layer
- RIL Worker Thread

Gonk
- RIL Proxy
- Vendor-specific RIL Daemon
- Baseband
- Linux Kernel

Gaia
- Gecko
- Gonk

- RIL Daemon uses parcels for high-performance IPC transport of data and object references.

RIL Worker: Heavy Lifting in JavaScript

- The RIL Worker is the central component of Firefox OS for enabling the high-level JavaScript API to access GSM-related functions (e.g., Web Telephony API).
- It communicates with the RIL Proxy using the IPC thread and uses the SpiderMonkey JSAPI to link between C++ and JavaScript functions.

- **Reception** – The handler receives the incoming parcel in a circular buffer and invokes the method corresponding to the unsolicited response type. The response methods may communicate with the high-level API using DOM messages (postMessage()).

```
RIL[UNSOLICITED_RESPONSE_CALL_STATE_CHANGED] = function UNSOLICITED_RESPONSE_CALL_STATE_CHANGED() {
    this.getCurrentCalls();
};
```

- **Transmission** – Request methods create a parcel and place it into the circular buffer for outgoing parcels where it will be fetched from the IPC thread.

```
getCurrentCalls: function getCurrentCalls() {
    Buf.simpleRequest(REQUEST_GET_CURRENT_CALLS);
}
```

```
simpleRequest: function simpleRequest(type, options) {
    this.newParcel(type, options);
    this.sendParcel();
}
```
Example: Active Calls State Machine

The RIL object maintains detailed information about every call in RIL.currentCalls[i].

Web Apps

- **Hosted app** – Website plus manifest file (MIME: application/x-web-app-manifest+json)

- **Packaged app** – ZIP archive with all content (including manifest)

- Apps can have one of three **security types** with varying Web API permissions.

<table>
<thead>
<tr>
<th>Security Type</th>
<th>Trust Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certified</td>
<td>Highly trusted</td>
<td>System apps approved by OEM</td>
</tr>
<tr>
<td>Privileged</td>
<td>Trusted</td>
<td>Approved by a marketplace</td>
</tr>
<tr>
<td>Web</td>
<td>Untrusted</td>
<td>Regular web content (online or installed)</td>
</tr>
</tbody>
</table>

- Privileged and certified apps **must** be packaged and signed.
Web App Permissions

- **Permissions** – By default, apps have the same permissions as websites. Any other permission needs to be explicitly enumerated by the manifest.

```javascript
permissions: {
  contacts: [
    { description: "Friends to send your awesome highscores to.";
    access: "read";
  ];
}
```

- Implicit permission – Granted by successful code review (e.g., for Socket API).
- Prompted permission – Ask user to deny or allow at runtime.

**Caveat:** Permissions are only updated when the web app is installed or updated through the API or when Firefox OS is run for the first time (after a Gecko update). For incremental changes, either make reset-gaia or, on the device, remove /data/b2g/mozilla.

- **Same-origin policy** – Requests will only succeed if they are made to the same origin that served the original web app.

- **systemXHR permission** – Make HTTP requests without any origin restrictions.

```javascript
var xhr = new XMLHttpRequest({mozSystem: true});
```

...and add mozSystem attribute

Web App Sandboxing

- **Sandboxing** – Web app data (e.g., cookies, local storage, permissions) are isolated from any other web app, even if their iframes point to the same origin.

<table>
<thead>
<tr>
<th>Web App A</th>
<th>Sandbox A₁ ≠ Sandbox A₂ ≠ Sandbox A₃</th>
<th>Web App B</th>
</tr>
</thead>
<tbody>
<tr>
<td>my.domain.com</td>
<td>my.domain.com</td>
<td>my.domain.com</td>
</tr>
<tr>
<td>&lt;iframe&gt;</td>
<td>&lt;iframe&gt;</td>
<td>&lt;iframe&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>No interdependencies via third-party websites</td>
<td>Redundant local data</td>
</tr>
<tr>
<td>No permissions are shared to prevent attacks</td>
<td>Unable to share local data between apps</td>
</tr>
</tbody>
</table>

Apps cannot open each other using iframes, not even via the app://app-id scheme.
### Web APIs for Certified Apps

<table>
<thead>
<tr>
<th>Web API</th>
<th>Description</th>
<th>W3C Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telephony</td>
<td>Send and receive phone calls</td>
<td>Editor’s Draft</td>
</tr>
<tr>
<td>SMS</td>
<td>Send and receive SMS messages and manage stored messages</td>
<td>Editor’s Draft</td>
</tr>
<tr>
<td>Idle</td>
<td>Get notifications when user is idle.</td>
<td>No</td>
</tr>
<tr>
<td>Settings</td>
<td>Set system-wide configurations that are saved permanently on the device.</td>
<td>No</td>
</tr>
<tr>
<td>Power Management</td>
<td>Turn hardware components on and off</td>
<td>No</td>
</tr>
<tr>
<td>Mobile Connection</td>
<td>Obtain details about current voice and data connection</td>
<td>No</td>
</tr>
<tr>
<td>WiFi Information</td>
<td>List of WiFi networks including signal strength and connected network</td>
<td>No</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>Access to Bluetooth hardware</td>
<td>No</td>
</tr>
<tr>
<td>Permissions</td>
<td>Allow settings app to manage all app permissions in a centralized location</td>
<td>No</td>
</tr>
<tr>
<td>Network stats</td>
<td>Monitor data usage and expose data to privileged apps</td>
<td>No</td>
</tr>
</tbody>
</table>

The Web APIs for certified apps are specific to the Firefox OS platform and they are far from standardization.

### Web APIs for Privileged Apps

<table>
<thead>
<tr>
<th>Web API</th>
<th>Description</th>
<th>W3C Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>TCP socket API with SSL support</td>
<td>Editor’s Draft</td>
</tr>
<tr>
<td>Device Storage</td>
<td>Manage files locally stored on the phone</td>
<td>No</td>
</tr>
<tr>
<td>Contacts</td>
<td>Manage contact information stored in the address book</td>
<td>Editor’s Draft</td>
</tr>
<tr>
<td>Browser</td>
<td>Implement a browser inside a web app</td>
<td>No</td>
</tr>
</tbody>
</table>

### W3C Technical Report Development Process – Maturity Levels

- **HTML5.1**: 2012
- **HTML5.1**: 2015
- **HTML5.1**: 2016
### Web APIs for Untrusted Apps

<table>
<thead>
<tr>
<th>Web API</th>
<th>Description</th>
<th>W3C Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vibration</td>
<td>Control device vibration for things like haptic feedback in games</td>
<td>Candidate Rec.</td>
</tr>
<tr>
<td>Screen Orientation</td>
<td>Get notification when screen orientation changes</td>
<td>Working Draft</td>
</tr>
<tr>
<td>Geolocation</td>
<td>Access to the current location of the end user</td>
<td>Candidate Rec.</td>
</tr>
<tr>
<td>Web App Installation</td>
<td>Application installation</td>
<td>Editor’s Draft</td>
</tr>
<tr>
<td>Battery Status</td>
<td>Information about battery charge level and if device is plugged in</td>
<td>Candidate Rec.</td>
</tr>
<tr>
<td>Alarm</td>
<td>Schedule a notification at a specific time</td>
<td>Working Draft</td>
</tr>
<tr>
<td>Time</td>
<td>Set and get time and timezone</td>
<td>No</td>
</tr>
<tr>
<td>Web Activities</td>
<td>Delegate an activity to another application</td>
<td>No</td>
</tr>
<tr>
<td>Web FM</td>
<td>Access FM radio</td>
<td>No</td>
</tr>
<tr>
<td>Web Payment</td>
<td>Initiate payments and refunds for virtual goods</td>
<td>No</td>
</tr>
<tr>
<td>IndexedDB</td>
<td>Client-side storage of structured data and high performance searches</td>
<td>Working Draft</td>
</tr>
<tr>
<td>Ambient light sensor</td>
<td>Device light sensor support</td>
<td>Working Draft</td>
</tr>
<tr>
<td>Proximity sensor</td>
<td>Device proximity sensor support</td>
<td>Working Draft</td>
</tr>
</tbody>
</table>

### Web App Installation API

- The Web App Installation API is already offered at the lowest trust level.
- Installation API available through `navigator.mozApps`.

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>install()</td>
<td>Install manifest for hosted web app</td>
</tr>
<tr>
<td>installPackaged()</td>
<td>Install packaged web app</td>
</tr>
<tr>
<td>getSelf()</td>
<td>Get App object of currently running web app</td>
</tr>
<tr>
<td>getInstalled()</td>
<td>Get list of all installed apps from current origin</td>
</tr>
</tbody>
</table>

- Marketplaces can use this API to install web apps on the smartphone.
- In general, **any** website can use this API for installing a web app on the smartphone:

```javascript
var checkIfInstalled = navigator.mozApps.getSelf();
checkIfInstalled.onsuccess = function() {
  if (checkIfInstalled.result) { ... } else {
    var install = document.querySelector("#install");
    install.onclick = function () {
      var installApp = navigator.mozApps.install(manifestURL);
      installApp.onsuccess = function(data) { ... ];
      installApp.onerror = function() { ... ];
  }
```

Asynchronous handlers
DOMRequest: The Asynchronous Helper

- Gecko provides the DOMRequest to represent an ongoing operation.
- Many Web APIs in Firefox OS return a DOMRequest object, e.g., mozApps.install().

<table>
<thead>
<tr>
<th>DOMRequest</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onsuccess</td>
<td>Called when the associated operation completes</td>
</tr>
<tr>
<td>onerror</td>
<td>Called when an error occurs while processing the operation</td>
</tr>
<tr>
<td>readyState</td>
<td>&quot;done&quot; or &quot;pending&quot;</td>
</tr>
<tr>
<td>result</td>
<td>The result of a completed operation</td>
</tr>
<tr>
<td>error</td>
<td>Error information</td>
</tr>
</tbody>
</table>

DOMRequest is a Mozilla-specific solution for handling asynchronous calls. The WHATWG has just introduced the similar concept of “Futures” in the DOM living standard.

- "A future provides asynchronous access to the result of an operation that is ongoing, has yet to start, or has completed."

Source: http://dom.spec.whatwg.org/#futures

Alarm API and Messages

- An app may want to schedule a message at a specific time, e.g., the calendar app. The Alarm API can wake up the phone and trigger particular behaviors at these specified time points.
- When the alarm goes off, an alarm message will be fired that will be dispatched to the associated message handler.

```javascript
navigator.mozAlarms.add(new Date("July 15, 2013 16:20:00"), "ignoreTimezone", { mydata: "some_data" });
```

Adjust to current timezone (honorTimezone) or as is.

- In the web app manifest, the messages property specifies the potential targets.
- mozSetMessageHandler() actually sets up the callback function that handles them.

```javascript
navigator.mozSetMessageHandler("alarm", function (msg) { });
```

- Other system messages can be handled, e.g., receiving a phone call (telephony-new-call).

The Alarm API can deliver messages only to the page that called add().
Notifications

→ Notifications are used to inform the user of events that require attention, e.g., missed calls or received SMS messages.
→ These notifications appear for short time at the top of the screen and can be viewed by dragging down the notification board.
→ Notification can be easily created using mozNotification, giving it a title and a description (and optional icon URL).

```
var notification = navigator.mozNotification.createNotification(
    "Title of the notification",
    "Description text of the notification");
```

→ There are events to react on clicking on a notification and when a notification is closed.

```javascript
notification.onclick = function() {...}
navigation.onclose = function() {...}
```

→ Notifications will only show up to the user if show() is called.

```javascript
notification.show();
```

→ Include desktop-notification permission in web app manifest.

Data Storage

→ Synchronous storage API: localStorage (also known as DOM Storage)
  → Simple API to store key / value pairs
  → Can only be used with strings
  → Blocks main thread, hence may cause terrible performance
  → Bad user experience: Frozen animations, choppy scrolling

```
console.log('Value of key:', localStorage.getItem('key'));
```

→ Asynchronous storage API: asyncStorage based on IndexedDB
  → API resembles DOM Storage, only enhanced with callbacks
  → Callbacks are placed on event loop
  → Startup overhead due to backlogging
  → Response times vary, may even be up to 200 ms for congested event loop

```
asyncStorage.getItem('key', function() {
    console.log('Value of key:', value);
});
```

“Never use localStorage. Not even as a last resort.” – Jonas Sicking, Software Engineer, Mozilla
A Very Special App: The System App

Encapsulate functionality in stand-alone modules.
- Expose single object with its methods.
- Keep modules as independent as possible.

Communication between system app and Chrome occurs using events.
- Example: Pressing a hardware button results in a Chrome event that is sent to the system app.

mozChromeEvent and mozContentEvent are only in place due to lack of proper API and will likely be removed in later versions of Firefox OS.

Events

Communication via DOM events:
- Event capture – Traverse from outmost node to innermost node, nodes may intercept and prevent propagation to children.
- Event bubbling – Traverse from innermost node causing the event to the outmost node.
  - stopPropagation() – Prevent further propagation of the event.
  - stopImmediatePropagation() – Listeners called in order of registration (addEventListener()), this call prevents the remaining ones to be called.

Example: Hardware Buttons
- hardware_buttons.js uses window.dispatchEvent() to send the “home” event to the window manager.
- lockscreen.js will stop propagation to window_manager.js if the smartphone is locked.
Example: Enhanced Lockscreen

- The net-m OS lockscreen is an example of how a Mobile Network Operator (MNO) may customize Firefox OS.
- The integrated weather widget retrieves its data via HTTP requests from a web service.
- Thus, the lockscreen manifest must specify the systemXHR permission to deal with different origin.
- The net-m OS quick launch bar enables easy access to commonly used services.
- Stickiness for prominent MNO services.

Testing Web Apps Using the Simulator

- For quick development without the device, use the Firefox OS Simulator.
- Available as a plug-in for Firefox Browser (r2d2b2g), bundling a B2G Desktop installation.
Testing Web Apps on the Smartphone

- Connect the Firefox OS smartphone via USB to the PC and use the Android Debug Bridge (adb).
- The Android logging system stores log messages in circular buffers, which can be viewed and filtered using logcat.
- Example: -s “GeckoConsole” filters low-level console output.
- Building the whole Firefox OS image for flashing is a time-consuming process and inadequate for trying out single web apps.
- As an alternative, only the particular web app being worked on can be pushed to the phone and started.

BUILD_APP_NAME=app_name make install-gaia

Developing For Fun or For Profit?

Source: https://marketplace.firefox.com/
Firefox OS Web Payment Architecture

Developers

App 1 (Server)  ...  App n (Server)

Mozilla

Firefox Marketplace

Developer Hub

Firebox Marketplace Developer Hub

For inclusion in the marketplace, developers need to register their web apps at the Firefox Marketplace Developer Hub.

Privileged apps will undergo a stringent review process.

Application Key and Secret

- The application key is used to identify the application in JWT requests.
- To ensure authenticity and integrity, every JWT request is cryptographically signed with the application secret.
- Both are issued by the Developer Hub.

Firefox Marketplace Developer Hub

Firefox Marketplace

Web Payment Provider

Firefox Marketplace

Web Payment Provider

Bango

PayPal

Mozilla

Firefox Market-
place

Developer
Hub

Notifications

Handset

App 1 (Client)  ...  App n (Client)

Marketplace Manifest

mozPay()

App 1 (Client)

App n (Client)

Mozilla

Firefox Market-
place

Developer
Hub

Notifications

Handset

App 1 (Client)  ...  App n (Client)

Marketplace Manifest

mozPay()

App 1 (Client)

App n (Client)
High-Level Web Payment Flow

- **Subscriber**
  - Purchase request
  - Create and sign JWT
  - Show payment UI
  - Confirm purchase

- **Web app (server-side)**
  - Purchase request
  - Signed JWT
  - Invoke payment API, check whitelist
  - Purchase request with JWT
  - Verify JWT signature
  - Show UI in trusted dialog
  - Confirm purchase
  - Append transaction ID to JWT and sign JWT
  - Postback
  - Verify JWT signature
  - Transaction ID
  - Process payment
  - Release content
  - Already charge customer although payment flow not complete.

- **Web app (client-side)**
  - Single sign JWT
  - Initialize payment
  - Transaction ID
  - Append transaction ID to JWT and sign JWT
  - Success

- **Web payment provider**

- **Payment service provider**

Developer's Guide to Web Payments

**Server-Side Logic**
- JWT signing and verification.
- Postback and chargeback handlers.
  - Signed POST request with a Mozilla transaction identifier (transactionID) must be verified.
  - MIME type is application/x-www-form-urlencoded and JWT can be found in notice parameter.
  - Plain text HTTP response with status code 200 OK and containing only Mozilla’s transaction identifier is expected.
  - Release content to user after successful postback and deactivate content after chargeback.

**Client-Side Logic**
- Render payment button.
- Generate JWT and send to server for signing.
- Call navigator.mozPay() with signed JWT.
- Assign success and failure callbacks to DOMRequest.
- Poll status from server.

JWT signing and verification must occur on server to hide the application secret key for fraud prevention.
Web Pay accepts payments from Gecko, supplies the payment screens inside the trusted dialog, and implements the transaction logic.

Solitude stores application details from Developer Hub and implements the API provided by the Payment Service Provider.

The Web Payment Provider needs to be whitelisted on handset.

The Web Payment Provider reference implementation is publicly available on GitHub.

Reference implementation: https://github.com/mozilla/webpay

Benefits of Firefox OS

- Mobile network operators have a fully customizable platform at hand where they can integrate their own services using their preferred business models.
- Developers can use web technologies for developing smartphone apps and distribute them on any channel they prefer.
- While these motivations are all nice and understandable, they are all secondary!

Would you buy a phone because the network operator can implement his favorite business models?
Would you buy a phone because apps are written in HTML5?
Why would you buy a particular phone?

Design and build features that set Firefox OS apart from the competition. Not only operators and developers, but first and foremost you and me must love it.
Challenges

→ With its huge dependency on the web, mobile network operators must provide a network infrastructure where data is not the bottleneck. Are the initial markets ready for that yet?

→ Will the limitation of data plans threaten the whole user experience? The ubiquitous web on mobile is as good as your available bandwidth. And what about roaming?

→ Being launched in emerging smartphone markets only, will Firefox OS get the reputation of a low-performance, entry-level phone? Will high-end iOS and Android users adopt?

→ Packaged apps are distributed in clear text and can be easier reverse-engineered than binary applications. Could this deter developers from developing sophisticated apps? Do you still want to do the update mess?

→ Huge content providers may be reluctant to support the platform due to the lack of standardized web-based Digital Rights Management? The W3C Encrypted Media Extensions could be the first step into the right direction.

→ While the web may indeed be the platform, Gecko is not the only vehicle. Most of the Web APIs used in Firefox OS are far from being standardized. Will the fragmentation of mobile browsers and Web APIs become the next hassle for web app developers?

→ How can people monetize web apps in general? The mozPay() API is a proof of concept for web payments, however tied to a single platform. The W3C Payments Taskforce is still in its infancy.