Chromium-based solutions to implement the GENIVI browser

Author: Jacobo Aragunde Pérez
Date: April 2015

Table of Contents
Introduction........................................................................................................................................2
Chromium........................................................................................................................................3
Crosswalk.......................................................................................................................................5
Chromium Embedded Framework (CEF).......................................................................................7
Summary.......................................................................................................................................8
Introduction

This document describes different solutions to build a browser that fulfills the requirements specified by GENIVI. The three options analyzed are:

- A solution based directly on Chromium code.
- A solution built on top of Crosswalk.
- A solution using the Chromium Embedded Framework.

This is not meant to be an exhaustive list, other potential solutions could be identified.

Summarized requirements

The analysis revolves around the fulfillment of following requirements:

- The browser engine should support the latest HTML5 standards.
- A browser nowadays is more than an HTML renderer; there are features that will be required like bookmarks, history, tabs and tab management, incognito mode, etc. A UI is instrumental to provide access to these features.
- The GENIVI browser may be used without an UI; in that case, it should provide an API defined by GENIVI providing access to all browser features, accessible through DBUS.
- The solution should be portable and work on different Linux-based stacks on top of ARM or Intel boards.
- Finally, while this point is subject to discussion, it seems that current and future embedded stacks are relying on Wayland as their solution for graphics, so the browser should be able to run on Wayland to be considered a truly portable solution.
Chromium

Chromium is the open-source basis of Google's Chrome products: browser and OS.

Design principles

Chromium is a vertical solution comprised by many layers: user interface, multiprocessing, web content sandboxing, window management and graphics rendering among others. It is designed to use a minimum number of external dependencies, so it can run on a light stack on its OS mode and be more easily portable as a browser.

Technical overview

The different layers in Chromium, as in any vertical solution, define APIs to interact with each other. For our purposes, there are three APIs that should be taken into account:

- Ozone is a platform abstraction layer that is used for low level input and graphics.
- The Blink API encapsulates the web engine functionality on a single-process architecture.
- The Content API adds multiprocess and other advanced features on top of Blink. Renderer processes are kept separated from the browser process and from each other.

Chromium development is focused on the product, in general it can be said there is no official support for third party Chromium-based solutions, e.g. no API compatibility is guaranteed across releases.

Description of the solution

An implementation directly based on Chromium could count with all the features in the browser and full flexibility to use them at the cost of increased maintenance complexity. Any patches required to implement the GENIVI browser on top of Chromium would unlikely to be accepted upstream.

Browsing operations would be implemented interfacing Chromium's Content API, while browser services like history, bookmarks or incognito which are not part of it should be interfaced directly through any internal APIs used in Chromium. The main problem with this approach is that these APIs are subject to changes at any time because they are not meant to be used by external developers.

As for the user interface, Chromium's own should be hidden and the only way to do it is patching the UI code, because there is no toolkit used to build it.

Chromium is officially supported on Intel-based Windows, Mac OS X and Linux with X11. Building on top of ARM devices seems to be possible, and there exist recipes to build it on ARM embedded devices based on OpenStack.
Finally, Wayland support would be achieved with the proper implementation of Ozone. Ozone-Wayland is a project aiming for this, it provides a good level of functionality although it is not completely finished. It is developed mainly by Intel.

**Considerations about the community**

Chrome is nowadays the most popular web browser, and its code base is being reused in a lot of different ways. Nonetheless, the focus of Google, who basically controls the Chromium project, are the products derived from it: the browser and Chrome OS.

For that reason, contributions that contribute the the project goals are likely to be accepted, for example those that improve the portability or stability of the code base; but that would definitely not be the case for the contributions that would form the core of the GENIVI browser.
Crosswalk

Crosswalk is an HTML application runtime based on the Chromium code. It is available for Android as an embeddable webview container and for Tizen as the system-wide application runtime.

Design principles

Crosswalk aims to provide a web runtime to develop applications based on web technologies instead of native toolkits, replacing the browser concepts with application development ones. Crosswalk provides interfaces for communication with the hardware and other applications. Chromium's multiprocess architecture is wrapped by Crosswalk's own, focused on applications and services.

Technical overview

The following analysis will focus on Crosswalk for Tizen. Crosswalk usage as a webview for Android is a very specific use case for this platform, and it would be very difficult to port as it is mostly implemented in Java and relies on Java code from the Android version of Chrome.

As explained in the previous section, Crosswalk is intended to run applications and not web pages. For that reason, concepts like the browser, pages, tabs, etc. are hidden and the internal API is designed around applications, each one with a window, and listeners on application events like termination, hide, suspend, etc.

Crosswalk reuses and adapts the multiprocess model of Chromium to its needs. It currently overloads the original browser process to act like a **system service**, taking care of the status of the applications and listening to user events to launch new applications, install and remove them, etc. User input is generated by a set of scripts that use DBUS to communicate with the system service.

It has been **announced** that the process model is going to be redesigned to improve the isolation of applications preventing them from sharing the browser process. DBUS usage will be removed in that new design and replaced with another IPC mechanism that has not yet been determined.

Description of the solution

Crosswalk system daemon would be modified to listen to DBUS and react opening new windows, loading pages, etc. In the current design, the existing DBUS interface may be reused but that is expected to change and should be implemented from scratch.

The main challenge is the lack of correspondence between the browser and application concepts; that causes practical problems when trying to re-adapt Crosswalk internal application APIs to a browser workflow. A browser implementation cannot completely rely on Crosswalk internal APIs, since they make assumptions about the applications life cycle that cannot be translated to a browser. An example is that an application is not expected to be reloaded, nor getting its process reused to
load a different URL; both operations result on crashes.

Some browser features, like history or bookmarks, seem to technically present in the code base but not easily accessible. They are not required in Crosswalk by design so they are subject to removal in the future, with the risk of being lost and required to reimplement. There is no UI in Crosswalk either.

Crosswalk for Tizen is officially supported on Intel boards, but Tizen can technically be built for ARM so that would include Crosswalk too. Yocto may be used to build Tizen but it is only tested on the Intel architecture. Regarding Wayland support, Crosswalk is an integral part of Tizen and as such it is already supporting Wayland through Ozone-Wayland; the latter project seems to be very closely related with Crosswalk, both supported by Intel directly.

Finally, it is worth mentioning that starting a project on top of Crosswalk at this moment could be risky, because we do not know the scope of the modifications that would be required to adapt to the announced design changes.

**Considerations about the community**

Crosswalk is a relatively recent project, born in 2013. To this date, it has not managed to attract many users, and its success as a runtime for HTML5 applications is very tied to Tizen's which has not been widely adopted either. At the same time, the contribution rate on Crosswalk seems to have shrunk in the current year.

A browser solution based on Crosswalk would consist on a customization of this piece of software and it is not likely that all the patches that form it would be valuable for Crosswalk codebase due to the diverging focus of the projects; after all, Crosswalk is trying to build a web runtime and not a browser. One specific example would be the DBUS API, which nowadays exists in Crosswalk but is planned to be removed; it would be difficult to get a new DBUS API accepted into the project again.
Chromium Embedded Framework (CEF)
The Chromium Embedded Framework (from now on, CEF) is a simple framework for embedding Chromium-based browsers in other applications.

Design principles
CEF is intended to provide a stable API for development of applications with embedded browsers. All browser abstractions are preserved, and the multiprocess architecture of Chromium is preserved and properly interfaced.

Technical overview
CEF wraps Chromium’s Content API providing a stable API to all its features and default implementations for the Content API delegates for the supported platforms. Applications written with CEF are insulated from the implementation details of Chromium and Blink, but still optionally have a fair amount of control on the browser workflow like resource loading, navigation, networking or JavaScript execution.

Description of the solution
A browser based on CEF would be an independent application that would incorporate CEF as a library and use its API. This approach will be very easy to maintain, it would only require periodical replacement of the CEF library with a minimal set of changes in the API which would be documented in the corresponding release notes.

Browser features from layers above the Content API are not present in CEF, like the history, bookmarks or incognito, so they would have to be implemented in the browser side. Being a library, there is no UI provided either.

Regarding portability, CEF is officially supported only on Intel-based Windows, Mac OS X and Linux with X11. Building on top of ARM devices and using other Ozone backends seems to be possible but requires changes inside CEF code base. There is a patch that can be used as a starting point but the connections with Ozone-Wayland have to be implemented. The patch and further developments upon it should ideally be split and merged into CEF codebase, otherwise the maintenance cost of this solution would be bigger and the main advantage of this solution would be gone.

Considerations about the community
The main risk with CEF is that, despite having a long life – the project was born in 2009, less than a year after Chrome was first released – and a wide user base, it still relies on Marshall Greenblatt, the project founder, who still authors most of the commits on the core project. On the bright side, any patches to CEF with the goal of improving its portability would likely be accepted as long as they meet the project quality standards.
Summary
Closing this report with a comparative analysis of the solutions for some key points.

HTML5 support
All the options presented are based on Chromium which provides cutting-edge standards support.

Browser features
Chromium is the most complete solution with regard to features; of course, that was expected because Chromium is exactly a browser. Crosswalk removes or hides browser features to build an application runtime instead, while CEF is only meant to be a library and removes any features not needed for this purpose.

GENIVI API
The implementation of a DBUS API would have to be done from scratch for any of the options. The DBUS code that exists in Crosswalk right now should not be used because it is expected to be removed.

Portability
There are no straightforward solutions for portability; any option would likely require time and experience to achieve it. Since Chromium is mainly targeting Intel-based systems, so any solution derived from it would likely have issues to be ported to ARM, specially given the broad diversity of SoCs available.

Wayland support
Crosswalk for Tizen comes already integrated with Ozone-Wayland, being the best option regarding Wayland support. Chromium is designed to be easily built with any Ozone backend, it would be the second best option. CEF would require some development to have a stable Linux-Wayland backend, which could become an officially maintained backend in the future.

Maintainability
Solutions based on bare Chromium or Crosswalk would likely be forks of the base project. The maintenance of a fork can be arduous, specially when the original project has a big codebase and a fast pace of changes, which is the case for Chromium. A browser based on CEF would be an independent project, much easier to maintain.

Another factor for maintainability is the popularity of the original project; if a project lacks resources forks would be forced to do more maintenance tasks; that is nearly impossible to happen to Chromium, but CEF's dependency on one person is a risk, while Crosswalk has not yet proven to
be a long-run project.

Summary table

<table>
<thead>
<tr>
<th></th>
<th>Chromium</th>
<th>Crosswalk</th>
<th>CEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Browser features</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GENIVI API</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wayland support</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintainability</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conclusions

Unfortunately, the choice of one of the presented options largely depends on the weight of the different goals to be achieved with that project and its specific needs. Chromium is the most complete solution regarding features, although difficult to maintain; Crosswalk is very interesting if the stack of choice is based on Tizen but there are uncertainties about its architecture; finally CEF would be the most maintainable solution but requiring more work to enable it on Wayland.

It is also worth mentioning there are solutions that are not part of this evaluation report that could be better suited depending on the scenario. While WebKit has lost commercial importance, it is still an active project with a very flexible architecture.