



Apertis

WebKit Clutter

Design

Author:	Gustavo Noronha Silva
Contributors:	Tomeu Vizoso
Version:	1.1.2
Status:	Final
Date:	17 November 2015
Last Reviewer:	Ekaterina Gerasimova

This design was produced exclusively using free and open source software.

Please consider the environment before printing this document.

DOCUMENT CHANGE LOG

Version	Date	Changes
1.1.2	2015-11-17	<ul style="list-style-type: none">• Update project name to Apertis• Replace Secure Automotive Cloud with Apertis• Fix links to wiki.gnome.org• Delete obsolete document properties• Improve language
1.1	2014-12-15	<ul style="list-style-type: none">• Updated to new template
1.0	2013-05-15	<ul style="list-style-type: none">• Fixed link to Collabora's WebKit Clutter site• Fixed typo• Made design final
0.4	2013-03-08	<ul style="list-style-type: none">• Improved notes on upstreaming• Adapted several chapters to account for the fact that many features have already been developed• Mention the main chapter, 5, after explaining why investing in different cairo backends is not likely to be a good option right now in section 5.1• Add section on customization (4.1)• Added section about black or white listing of applications (8.1)• Added chapter about audio streams management (9)• Added chapter about rendering of non-web documents, such as Microsoft Office documents and PDF (10)• Added new chapter about scheduled and potential future work (12), with sections about Web runtime (12.1), General performance (12.2) and JavaScript performance (12.3)
0.3.1	2012-05-11	<ul style="list-style-type: none">• Updated title and file name to follow Document Naming Scheme
0.3	2012-05-03	<ul style="list-style-type: none">• Integrated Travis review• Added information about security maintenance of a WebKit port• Integrated Martin Barrett's review
0.2	2012-04-30	<ul style="list-style-type: none">• Section about contextual zoom
0.1	2012-04-25	<ul style="list-style-type: none">• Started• Imported WebGL chapter from Clutter/MT design

Table of Contents

Document Change Log.....	2
1 Introduction.....	4
2 Clutter port maintenance.....	5
2.1 Notes on upstreaming.....	5
2.2 Bug fixing.....	5
2.3 Security maintenance.....	5
3 Tracking Clutter improvements.....	7
4 Event handling overhaul.....	8
4.1 Customization.....	9
5 Tiled backing store with improved scrolling performance.....	10
5.1 Different Cairo backends.....	10
6 WebGL support.....	11
6.1 Requirements.....	11
6.1.1 WebGL implementation.....	11
6.1.2 Security.....	11
6.2 Approach.....	12
6.3 Risks.....	12
7 Contextual zoom.....	14
8 Script watchdog.....	15
8.1 White-listing and black-listing.....	15
9 Audio streams management.....	16
10 Rendering of non-web documents.....	17
11 Rendering of HTML form elements and other in-page UIs.....	18
12 Scheduled and potential future work.....	19
12.1 Web runtime.....	19
12.2 General performance.....	19
12.3 JavaScript performance.....	19

1 INTRODUCTION

The browser is an important piece of the Apertis system. The WebKit Clutter port has been started and maintained by Collabora for the Apertis system.

2 CLUTTER PORT MAINTENANCE

As discussed earlier, the Clutter port of WebKit will require maintenance and frequent merges from upstream to both keep the maintenance burden low and get new features. Collabora intends to do frequent merges of upstream's trunk followed by creating a new deb package for integration and testing.

2.1 NOTES ON UPSTREAMING

Whether to upstream and the work involved in upstreaming have been discussed several times since the project started.

Not upstreaming causes difficulty in keeping up with upstream and impacts long-term maintainability. However, upstreaming itself is a significant chunk of work that will cause some friction with the WebKitGTK+ team and requires long-term commitment by the people involved.

With a possible GTK+ and Clutter merger¹ looming in the horizon, Collabora currently believes upstreaming the Clutter port as a whole is not a worthy course of action right now, and intends to offset the risks involved with not doing it by tracking upstream closely by merging often.

The fact that the large majority of the changes done to create the Clutter port are around the core components means the risk is more manageable, as well.

Collabora has been pushing any changes that make sense in core components upstream, such as fixes to the shared WebGL code and the libsoup HTTP backend.

One additional piece that would be useful to have upstream is the Clutter-based accelerated compositing framework. It can be used by the WebKitGTK+ port and Collabora believes it would be very useful for the Apertis project to have it already upstream when the Clutter/GTK+ merger comes through. At the moment, Collabora is pushing the AC work upstream as an R&D investment.

2.2 BUG FIXING

Collabora has two main bug trackers for the WebKit Clutter port at the moment: the Apertis project's bugzilla, and a public one hosted by Collabora². Collabora will work on bug fixing for WebKit Clutter features developed for the Apertis project and for WebKit Clutter bugs in general, as feature development and other tasks allow.

2.3 SECURITY MAINTENANCE

One important aspect of browser engines is tracking security fixes after a stable release. The WebKit project itself does not have releases, and those are left for

¹ At the moment it seems like there are two courses of action being considered in the GNOME community: either Clutter becomes the base on which GTK+ layout and animations is built on, or GTK+ grows the same features Clutter provides itself, it is not clear which path will be followed, experiments to assess the pros and cons are currently under way.

² <http://webkit-clutter.collabora.co.uk/>

each port to do. That means each port is also responsible for doing security maintenance. Currently the WebKit Clutter port and its sibling WebKitGTK+ lack security support, specially long term.

Security support is a lot of work and is not currently part of the scope of this project. The WebKitGTK+ team has been trying to improve the situation and if they are successful the Clutter port could use their work by using the WebKitGTK+ stable branch for deriving its releases from. This conflicts with the idea of merging often from trunk, but WebKitGTK+ releases every 6 months, so it might be possible to track their releases while keeping fresh at the same time.

3 TRACKING CLUTTER IMPROVEMENTS

Clutter 1.10 has brought with it a large reorganization of the classes and some additional features. As work progresses in the Apertis project and newer versions of Clutter are integrated, those features will be usable for better supporting accelerated compositing functionality.

An example is the number of replays an animation should have. Previously, that had to be implemented manually in the accelerated compositing code, but this has been since added as a feature to Clutter itself. Note that we do not benefit from these improvements just by upgrading the version of Clutter we ship: development work is required to adopt the new functionality.

4 EVENT HANDLING OVERHAUL

Event-handling has been an ongoing issue which Collabora has revisited continually since the beginning of the port. This problem is complex due to the difficulty of balancing the requirements of the Mx scrolling widgets, web content behavior, and the Apertis system as a whole. The following elements need to be considered:

- Main frame touch-based kinetic scrolling
- Scrolling of in-content scrollable elements such as iframes and overflown divs
- HTML drag and drop support
- Nearest interesting target click (“lazy click”)

Collabora believes that bringing this solution into the web engine is the most-manageable way to solve this complexity. This means all event handling requirements, including scrolling, would be implemented inside WebCore. The main advantage of this is allowing the component that has the most knowledge about the details of the content make the decisions regarding events while at the same time allowing for sharing of code with other ports.

A recent trend in the WebKit development world is recognizing that the new event handling requirements brought by the mobile and touch screens are becoming largely mainstream. The number of custom implementations has been going up and their complexity as well. Desire for greater code sharing and for making those use cases first class citizens has pushed developers to make those built-in features of the engine. Collabora believes the work on event handling for the Apertis project can both take advantage of that new trend and contribute to making it happen.

The first step carried out by Collabora towards this goal was to move responsibility for performing scrolling and panning from Mx scrolling widgets to WebCore, reusing existing infrastructure for gesture handling, and leveraging Clutter's gesture recognition engine. More work was done after that to add the same configurability allowed by the Mx widgets. This work has been provided in the **(2012Q3)** release, and received improvements and fixes during the **(2012Q4)** development cycle.

A final important requirement that is useful to have built into WebCore is the feature called lazy click, which does a search for interesting nodes around the point that got touched to account for the coarser accuracy of fingers compared to mouse pointers. This feature has been delivered in the **(2012Q4)** release.

4.1 CUSTOMIZATION

There has been some interest in allowing customizability of the scrolling behaviour.

Scrolling of the main page content may be forced (as opposed to scrolling in-page scrollable content) by using two fingers to scroll, while single-finger dragging

would use the current context-aware behaviour; other OEMs may want the opposite, or just the context-aware behaviour.

Another potential OEM customization would be to customize the bouncing animation – it's bounce time, or other behaviours.

Collabora believes providing customization to this extent would make the code significantly more complex and hard to maintain. It would also probably deviate more from upstream and make the frequent merges harder to do. In addition, it is worth considering the impact on the usability of the system by people who drive different brands that ship Apertis, since the slightly different behaviour depending on the OEM may be very confusing.

Nonetheless, customization of these behaviours by editing the code, although not trivial, is not too hard to perform, so Collabora recommends this approach should be used should any OEM want to change these behaviours.

5 TILED BACKING STORE WITH IMPROVED SCROLLING PERFORMANCE

As part of the Accelerated Compositing work, the WebKit-Clutter widget has been turned into a container that can have arbitrary actors added to it. This infrastructure was used to build a backend for the tiled backing store that is much more efficient, having the tiles be actors, which are then placed directly on the stage.

This change has increased scrolling performance for slow scrolling (often referred to as panning), and for faster scrolling, with improvements of up to 100% in FPS, with up to 50% reduced CPU usage, even with the system under stress.

5.1 DIFFERENT CAIRO BACKENDS

Another way to improve painting operations' performance is by having them not go through a Cairo image surface on its way to the Cogl texture. Intel has started a Cogl backend that paints directly to the texture. During the work on the Clutter port of WebKit, there was discussion with Intel about using this backend. The news Intel brought forward was that the backend was still a work in progress and any serious testing would not be done for a while. So, Collabora believes that at this point in time this backend is not a viable option.

Another option would be to use a different GL or GLES based backend such as Coiro-GL. This would be challenging because mixing GL/GLES with Cogl (and thus Clutter) usage is not yet supported, but the work done towards making WebGL viable could potentially also be used as a base for this.

Unfortunately, Coiro-GL is still an experimental backend and has performed poorly on benchmarks that have been run. A benchmark ran recently by a Cairo developer has shown the GL backend to be consistently outperformed by the baseline image backend which renders purely using the CPU³.

The results might differ depending on the hardware – in particular, it might be that ARM's CPU underperforms the Vivante GPU, but there is no substantive reason to believe so at the moment. Linaro has indicated the desire of improving GLES integration with Cairo and validating its performance⁴, but so far results are not forthcoming.

At this point Collabora believes using a Cairo backend based on GL/GLES is not a clear win, and so would advise against it unless some other party has an interest in driving the effort, which is bound to require significant resources and effort.

Work that resulted in a very significant improvement in scrolling performance has been done by Collabora for the **(2012Q3)** release. This work used Clutter actors to serve as the tiles used by the backing store, as explained in chapter 5.

3 <http://people.freedesktop.org/~ickle/snb+gl-image-1.12.png> see <http://ickle.wordpress.com/2012/03/28/cairo-1-12-let-the-releases-roll/> for the whole blog post
4 <https://wiki.linaro.org/WorkingGroups/Middleware/Graphics/Specs/1111/engr-components-cairo>

6 WEBGL SUPPORT

WebGL⁵ specifies a way of using an API very similar to Open GL ES 2.0 from web pages. This allows web developers to deploy to browsers applications that used to be possible only on the desktop. Examples of such applications are realistic games, complex data visualization, geographic visualization and 3D modeling⁶. It also allows the GPU to be used for tasks that were already possible using the HTML 2D Canvas API⁷ but were too slow in some devices or consumed too much battery power⁸.

Through the specification is very new, most browsers already support it, though often support is not complete or addresses an earlier draft. WebKit already has fairly good support for WebGL, and it continues to be improved by Safari, Chromium and GNOME developers.

Browsers usually just bridge JS calls to WebGL methods to their OpenGL or OpenGL ES equivalent, but in the case of WebKit-Clutter this cannot be done as easily because it is not currently possible to safely use raw GL calls in a Clutter application. This design will address the changes that are needed in Cogl so that is possible.

6.1 REQUIREMENTS

6.1.1 WEBGL IMPLEMENTATION

WebGL support in WebKit is quite complete, though not all ports have implementations of the same quality. Collabora has implemented the WebGL backend for the Clutter port so it is now on par with the WebKit-based browsers that have best support for WebGL as of February 2013.

Among the code that the Clutter port reuses from within WebKit is the ANGLE shader translator. Though the WebGL shader language specification is very closely based on the OpenGL ES Shading Language, there are some differences due to security considerations. The ANGLE shader translator transforms WebGL shaders in the format that the OpenGL ES driver requires.

6.1.2 SECURITY

There is a good explanation of the security issues with WebGL implementations in a Khronos WebGL security article⁹. This section explains how WebKit addresses each of them:

- Undefined Behavior: All WebGL implementations have to implement the behavior from the spec and the conformance test suite tests it.
- Out of Range Memory Accesses: Same as above, the WebGL spec details

5 <http://www.khronos.org/webgl/>

6 http://en.wikipedia.org/wiki/Google_Body

7 <http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html>

8 <http://chrome.angrybirds.com/>

9 <http://www.khronos.org/webgl/security/>

the checks that must be done before calling the GL or GL ES implementation and these are tested in the conformance suite.

- Access to Uninitialized Memory: Same as above.
- Shader Validation and Transformation: Even though the Apertis platform uses GL ES 2.0 and thus its GLSL version matches that of WebGL, all shader code is passed to the ANGLE shader compiler¹⁰ which validates it accordingly.
- Denial of Service: If the GPU vendor implements the EXT_robustness¹¹ extension and makes use of it when a process uses too many resources, WebKit will gracefully reset the WebGL context avoiding a DoS situation.
- Cross-Origin Media: WebKit implements CORS¹² (Cross-Origin Resource Sharing) in all image and video elements and also to texture loading in WebGL.
- Blacklisting graphics drivers: This is not implemented in WebKit, but is not needed in platforms where the specific version of the GPU drivers is known and can be tested beforehand.

Because WebGL is a thin layer on top of the GL ES 2.0 API, the provider of its implementation has to take serious care of its security, because most security flaws will be exposed to web pages thus being exploitable remotely.

6.2 APPROACH

The clutter backend for WebGL is based on the GTK+ one. The bridging of most calls from JS to the underlying GL implementation is very similar, but the setup of the GL context is very different, as it is done through Cogl.

As mentioned before, it is not safe for Clutter applications to call GL directly. This is because Cogl makes the assumption that pipeline state is not changed by other parties. Thus, Cogl can reduce the amount of state switching and batch geometry calls between state changes for substantial performance gains. To support raw GL calls, Collabora added functionality to Cogl so an additional GL context can be created for those raw GL calls, leaving the pipeline state of the Cogl context untouched.

Also, an indirection layer is used that covers the whole GL ES 2.0 API to make sure WebKit uses the same GL library as Cogl.

6.3 RISKS

WebKit2 has moved to a split process model in which the UI runs in one process and each page of web content runs in its own process. In case it's decided to port WebKit-Clutter to WebKit2 for the Apertis project, Collabora will have to find a way to efficiently render WebGL canvases into textures that the UI process can

¹⁰ <http://code.google.com/p/angleproject/>

¹¹ http://www.khronos.org/registry/gles/extensions/EXT/EXT_robustness.txt

¹² <http://www.w3.org/TR/cors/>

present. The two main alternatives are either rendering to a frame buffer object (FBO) from the Web process and sharing that texture to the UI process, or serializing the GL commands and sending them from the Web to the UI process for execution.

7 CONTEXTUAL ZOOM

Contextual zoom is the feature found on most smartphones today that allows the user to double tap a part of the page so that the browser closes up on that part of the page, making sure the content you're interested in fits the whole width of the browser viewport. This makes for better reading (especially on multi-column websites). Some implementations, such as android's go a step ahead, and modify the width of the content to make the content even more readable.

This has to date usually been a browser feature rather than an engine feature. The same trend that recognizes the mobile scrolling requirements have also reached this one, though, and including something similar to this as shared code in WebCore makes sense.

The requirements are:

- Provide an API either inside WebKit-Clutter or as a separate library that allows zooming to a part of the page
- The zoom needs to be animated in a way that there is no discontinuity
- The API should provide enough flexibility to allow the browser to control the zooming animation

This feature has been implemented and delivered as part of the **(2012Q4)** release.

8 SCRIPT WATCHDOG

If a single block of JavaScript runs uninterrupted for a long time, the UI can become unresponsive and the CPU usage can starve other processes or waste significant battery capacity. WebCore monitors how long a block of JavaScript has been running and gives the chance to the browser to terminate it¹³.

Collabora added API to webkit-clutter for the browser to decide what to do in that circumstance, as part of the **(2012Q3)** release. For example, it could ask the user whether the execution of the block should be stopped, or could stop it right away to conserve resources.

Note, however, that testing this feature has proven to be a challenging endeavour. This indicates that there are some cases which are not caught by the watchdog. Depending on how this feature turns out on some real-world testing, it may make sense to invest in researching and improving the inner workings of the watchdog.

8.1 WHITE-LISTING AND BLACK-LISTING

There has been interest in maintaining a black-list of web applications (or pages) that misbehaved. That would be for the case in which the browser gets killed because it stopped responding and the scripts watchdog was not able to restore it to working, so that those web apps or pages are not loaded automatically upon startup causing the browser to go unresponsive again.

Web¹⁴ (codename Epiphany¹⁵), the GNOME web browser maintains a session file that stores information about all loaded pages, such as title, URL, and whether they are currently loading or not. If Web is quit unexpectedly, it will refuse to load any pages that were marked as still loading automatically. This same approach could be used by the Apertis web browser to not load those pages automatically or to create a blacklist database.

The problem with creating a blacklist from this is that it is quite likely it will hit false positives: if multiple tabs are loading, but only one causes the crash, the browser does not have enough information to detect which page caused the problem. In the future, if the browser is moved to WebKit2, using a web process per tab/app, this behaviour could be improved.

The white-list, on the other hand, would be used to enable applications to use lots of resources for a long time without getting killed by this infrastructure in what could be considered a false positive. A white-list can easily be implemented, keeping a list of applications that are allowed to go over the limits should be enough.

¹³ <http://trac.webkit.org/browser/trunk/Source/WebKit/gtk/WebCoreSupport/ChromeClientGtk.cpp#L365>

¹⁴ <https://wiki.gnome.org/Design/Web>

¹⁵ <https://wiki.gnome.org/Epiphany>

9 AUDIO STREAMS MANAGEMENT

Ensuring an audio stream that is currently playing is stopped when another, higher priority, stream is started is one of the requirements for the web browser. In discussions over email, Collabora has described how this functionality works when using PulseAudio in an email exchange back in May 2012¹⁶. Collabora believes this should be considered as a system-wide functionality and applied to all media applications, including the browser.

Support for pausing and unpausing streams when PulseAudio corks or uncorks a stream is being added to the upstream WebKit GStreamer backend for HTML5 media by Igalia¹⁷, which means Apertis will be able to take advantage of it when it is ready and merged. The work-in-progress patch sets the role of the stream so PulseAudio can know if that stream is for a video or for audio-only, and a policy plugin can then apply whatever rules are required.

¹⁶ See email with subject *Re: Queries regarding webkit*, sent in the morning of May 30, 2012; see also http://freedesktop.org/software/pulseaudio/doxygen/stream_8h.html#a14e698233ac2d246646651955ab0ec7b

¹⁷ https://bugs.webkit.org/show_bug.cgi?id=91611

10 RENDERING OF NON-WEB DOCUMENTS

Several kinds of documents that are not strictly web documents are available on web sites for viewing and download. Some of these types of documents, such as PDFs, have become so common that some browsers embed a viewer.

WebKit itself does not have support for rendering those documents and the WebView actor provided by WebKit Clutter actor does not support any kind of custom rendering. However, embedding a separate actor that uses a PDF rendering library to be shown may be considered instead of the WebView actor for PDF documents. The same goes for word processing documents, such as Microsoft Office files, for instance.

Mozilla has started an experimental JavaScript library called PDF.js¹⁸ which is able to render PDF files in a web page without the need for any kind of plugin. Collabora has not experimented with the library, but it is worth investigating its viability, particularly in terms of performance. If it turns out to meet performance requirements, it will likely be easier to integrate than platform PDF rendering libraries, unless there is already a working solution.

¹⁸ <https://github.com/mozilla/pdf.js/wiki>

11 RENDERING OF HTML FORM ELEMENTS AND OTHER IN-PAGE UIS

WebKit has support for custom drawing of form elements and other in-page UIs, such as media controls. Collabora has made most of the rendering available to be implemented in the toolkit library layer, and recently added some of the newer ones. Customized rendering of media controls is also one of the support/feature request topics. Collabora can help with advice, support or with the development of custom rendering or plumbing.

12 SCHEDULED AND POTENTIAL FUTURE WORK

12.1 WEB RUNTIME

There is interest in providing developers with a way to write applications using web technologies for Apertis. While this is out of the scope of this design, a small description of existing technologies and how they can be applied follows. Collabora can help in the future with a more detailed analysis of what works needs doing, specification and development of a solution.

WebKit Clutter supports roughly the same set of technologies as those supported by WebKitGTK+. That means Application Cache, Local Storage, IndexedDB (which replaced the now deprecated Web Database) are all supported, in addition to HTML5 canvas, media elements, WebGL, and so on.

A runner for web applications would ideally create a process for each separate application that will be executed, and use application-specific locations for storing data such as caches and the databases for the features described above – meaning it would not require any kind of special privilege, it would be a regular application. This also means permissions and resource limits can be set individually also for web applications.

The fact that more than one process would be executed does not mean a lot of memory overhead, since shared libraries are only loaded in memory once for all processes that use them. This would also have several advantages such as making managing applications permissions easier, and avoiding one application interfering with others.

The Apertis platform already ships one of the main components that would be required for integrating with platform APIs: the seed library. There has already been some experimentation with it, and Collabora has helped fix a few issues that came up. Collabora will continue to provide support through support requests, and is available should a more detailed consultancy be required to further detail and implement this.

12.2 GENERAL PERFORMANCE

Performance of the web engine in general will be looked at during the **(2013Q2)** release cycle, focused on load time and memory consumption, essentially. Graphics performance, including performance for HTML5 elements such as the 2D canvas will have to be postponed, since the lower level components that need to be released by FreeScale and Vivante are not working properly at the moment and working on improvements done to these areas for the intel reference hardware will not necessarily be portable.

12.3 JAVASCRIPT PERFORMANCE

In terms of JavaScript performance, WebKit Clutter is state of the art, since it employs the same JS engine used by Apple's Mobile Safari on the iPhone and iPad.

The engine has all of the features, including the “SquirrelFish Extreme” Just In Time (JIT) compiler¹⁹ - marketed by Apple as *Nitro*, the LLInt interpreter²⁰ for code for which the JIT would only add overhead, and DFG²¹. Any work here would require high expertise in compiler, JIT and interpreter technologies, as there are no low-hanging fruits for quite a while.

¹⁹ <https://www.webkit.org/blog/214/introducing-squirrelfish-extreme/>

²⁰ http://appleinsider.com/articles/12/03/01/new_low_level_javascript_interpreter_to_boost_webkit_performance_more_than_200

²¹ <http://2012.jsconf.eu/speaker/2012/08/29/javascriptcore-s-dfg-jit.html>