UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA DE
SISTEMAS AUDIOVISUALES

TRABAJO FIN DE GRADO

ESTUDIO SOBRE EL DESARROLLO DE APLICACIONES CON SOPORTE MULTIMEDIA EN PLATAFORMAS PARA MÓVILES BASADAS EN WEB

AUTORA: VERÓNICA ROMÁN LÓPEZ

TUTORA: FLORINA ALMENARES MENDOZA

Marzo de 2014

Título: *ESTUDIO SOBRE EL DESARROLLO DE APLICACIONES CON SOPORTE MULTIMEDIA EN PLATAFORMAS PARA MÓVILES BASADAS EN WEB*

Autora: *VERÓNICA ROMÁN LÓPEZ*

Tutora: *FLORINA ALMENARES MENDOZA*

La defensa del presente Trabajo Fin de Grado se realizó el día 04 de Marzo de 2014; siendo calificada por el siguiente tribunal:

Presidente: *IRIA MANUELA ESTÉVEZ AYRES*

Secretario *VÍCTOR ELVIRA ARREGUI*

Vocal *ALMUDENA LINDOSO MUÑOZ*

Habiendo obtenido la siguiente calificación:

Calificación:

Presidente **Secretario** **Vocal**

Agradecimientos

Me gustaría que estas líneas sirvieran para expresar mi más profundo y sincero agradecimiento a todas aquellas personas que con su ayuda han colaborado en la realización del presente proyecto, en especial a mis padres, Wenceslao y María Dolores, quienes hicieron todo en la vida para que yo pudiera lograr mis sueños, por motivarme y darme la mano cuando sentía que el camino se terminaba, a ellos siempre les estaré agradecida. Y a mi hermana, Aroa, pues sin su apoyo y asesoramiento nada de esto hubiese sido posible.

También quiero dedicar estas líneas a mi novio, Ángel, por darme ánimos para finalizar esta travesía. Especialmente por su comprensión y por su paciencia conmigo, estando ahí en todo momento, llenándome de ilusión cada día y haciéndome disfrutar a lo grande cada pequeño momento. Ya que siempre a preferido sacrificar su tiempo para que yo pudiera cumplir con el mío. Por su bondad y sacrificio, gracias por estar siempre a mi lado cariño.

A mis abuelas y abuelos que, aunque éstos últimos no se encuentran entre nosotros, sé que estarían muy orgullosos de mí, al igual que lo están mis abuelas, por haber finalizado esta etapa de mi vida. Por lo que siempre les agradeceré toda la sabiduría que me han proporcionado y que tanto ha influido en mi madurez para lograr todos los objetivos en la vida.

A mis compañeros de clase, por hacerme más amenas y llevaderas las largas jornadas en la Universidad, ya que sin sus buenos ratos y sin su ayuda todo hubiese sido más difícil.

A todos mis amigos y familiares, que siempre me han apoyado, porque ellos siempre me han hecho sentir diferente, única, y lo más importante: querida.

A mi tutora, Florina Almenarez Mendoza, por ofrecerme este Trabajo Fin de Grado y mostrarse siempre disponible para proporcionarme ayuda y facilitarme la consecución de los objetivos. Así como por la orientación, el seguimiento y la supervisión continúa que

ha tenido conmigo, pero sobre todo por la motivación y el apoyo recibido durante todo este periodo.

A mis compañeros de Mediaset España, por acogerme con tantísimo cariño y darme la oportunidad de poder aprender cada día junto a ellos y, sobre todo, por formarme tanto profesional como personalmente.

A todo el personal docente que durante estos años han moldeado lo que hoy soy gracias a su esfuerzo y dedicación, y a esta institución que me ha formado.

También quiero dar las gracias a Soledad Penadés, ya que gracias a ella he tenido un gran asesoramiento en todo lo referente a Firefox OS.

A todos ellos, muchas gracias.

Si quieres ser sabio, aprende a interrogar razonablemente, a escuchar con atención, a responder serenamente y a callar cuando no tengas nada que decir.

Johann Kaspar Lavater

No hay que confundir nunca el conocimiento con la sabiduría. El primero nos sirve para ganarnos la vida; la sabiduría nos ayuda a vivir.

Sorcha Carey

No esperes que lleguen las circunstancias ideales ni la mejor ocasión para actuar, porque tal vez no lleguen nunca.

Anónimo

Resumen

Tal vez podrían preguntarse si un nuevo sistema operativo para móviles tiene sentido en un mundo dominado por Google y Apple. En este Trabajo Fin de Grado se estudiarán los nuevos sistemas operativos que están surgiendo, tales como *Firefox OS*, *Tizen OS*, *Ubuntu Touch OS*, *Sailfish OS* y *WebOS*, los cuales han optado por sacar el máximo partido al navegador web a partir de aplicaciones basadas en *HTML 5*, *CSS 3* y *JavaScript* y, se verá como éstos comienzan a plantar cara a los dos sistemas operativos por excelencia de hoy día, Android e iOS.

Cabe destacar, que como primera toma de contacto con cada uno de ellos, se ha creado una aplicación básica como lo es el conocido "HolaMundo", la cuál ha sido desarrollada en todos estos entornos.

Pero principalmente, el estudio se ha enfocado a analizar las APIs multimedia que se han empleado en dos aplicaciones desarrolladas dentro del entorno *Firefox OS* (ya que se disponía de dispositivo). En donde en una de ellas se decide, desde un principio, que no acceda al *hardware* del terminal en ningún momento, al contrario que la otra, que sí deberá acceder a él. Posteriormente, estas aplicaciones serán evaluadas en aquellos entornos que más se asemejen y difieran de *Firefox OS*. De este modo, el estudio será más completo ya que se comparan ambas posibilidades de acceso en varias plataformas móviles, obteniendo una serie de resultados sólidos de donde se extraerán las conclusiones finales.

Indicar a modo de conclusión que lo interesante de estos sistemas operativos que se estudiarán, no es el sistema en sí, sino las aplicaciones. Las cuales se basan en tecnologías web y, por lo tanto, son mucho más fáciles de programar, más versátiles y accesibles.

Palabras Clave

Plataformas móviles, OS, sistema operativo móvil, Firefox, Tizen, Ubuntu Touch, WebOS, Sailfish, Web, Android, iOS, HTML 5, CSS 3, JavaScript, Qt, smartphone, dispositivo, hardware, APIs, multimedia, Firefox OS y tablet.

Abstract

Perhaps might wonder if a new mobile operating system makes sense in a world dominated by Google and Apple. In this Bachelor Thesis new operating systems are emerging, such as *Firefox OS*, *Tizen OS*, *Touch Ubuntu OS*, *Sailfish OS* and *be studied WebOS*, which have chosen to make the most from the web browser-based *HTML 5 applications*, *CSS 3* and *JavaScript*, and will look as they begin to stand up the two operating systems par excellence today, Android and iOS.

Remarkably, as first contact with each of them, has created a basic application is known as "HelloWorld", the which has been developed in all these environments.

But mostly, the study has focused on analyzing the multimedia APIs that have been used on two applications developed within the Firefox OS environment (as were available device). Where one of them is decided from the outset not to access the hardware of the terminal at any time, unlike the other, we do need to access it. Subsequently, these applications will be evaluated in environments that most closely OS and Firefox differ. Thus, the study will be more complete as both accessibility in various mobile platforms are compared, obtaining a series of strong results from which the conclusions are drawn.

Report Concluding that interesting of these operating systems to be studied, not the system itself, but the applications. Which are based on web technology and, therefore, they are much easier to program, more versatile and accessible.

Keywords

Mobile platforms, OS, Mobile OS, Firefox, Tizen, Ubuntu Touch, WebOS, Sailfish, Web, Android, iOS, HTML5, CSS3, JavaScript, Qt, smartphone, device, hardware, APIs, multimedia, Firefox and tablet OS.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción y objetivos

1.1. Introducción

Hoy en día, el ser humano se mueve constantemente. Se vive en un mundo en el que el estrés se alza como claro dominador, un mundo en el que la pérdida de un segundo puede ser vital en muchos sentidos, un mundo en el que el hombre pretende llegar a todas partes sin ser un dios...pero ante todo, se trata de un mundo en el que el ansia de satisfacer todos estos requerimientos nos ha llevado a reducir al hombre y a la máquina en un mismo ente como dos partes de un todo...

Los avances tecnológicos en los últimos años han impulsado a la sociedad actual a inclinarse con una confianza creciente hacia las distintas posibilidades de explorar servicios antes desconocidos; como es la aparición de nuevas generaciones y nuevas creaciones en distintos ámbitos.

La utilización de dispositivos móviles de tipo "*handheld*" en las organizaciones ha escalado posiciones en la jerarquía de soluciones de administración de datos; debido principalmente a la comodidad de transportarlas, a que su interfaz es más sencilla respecto de un PC, como así también la posibilidad de conexiones inalámbricas a redes de datos; pero, éstas tienen ciertas limitaciones como su capacidad de memoria; aún así estos dispositivos móviles son de gran utilidad y poseen cada vez más prestaciones [1].

Debido a que disponen de mayores servicios para el usuario, la tecnología móvil es un mercado en constante evolución que en los últimos años ha dado un auténtico "salto de gigante". Sin embargo, este sector aspira a hacerse aún más grande, introduciendo nuevos avances y evolucionando constantemente.

A continuación, en el presente proyecto, se realizará un estudio referente al desarrollo de aplicaciones con soporte multimedia a partir de plataformas móviles basadas en Web.

1.2. Motivación

La principal motivación por la que se ha elegido realizar este proyecto se debe a que los sistemas operativos móviles basados en Web que acaban de salir prácticamente al mercado, proponen algo que ni Android ni IOS pueden proponer, compatibilidad total entre ellos. Esta peculiar característica es lo que les hace interesantes, ya que tanto *Firefox OS*, *WebOS*, *Tizen OS*, *Ubuntu Touch OS* y *Sailfish OS* están basados de alguna manera en el formato estándar de *HTML 5*, lo que facilita el desarrollo de aplicaciones para ellos, de hecho, se puede desarrollar una aplicación y si se logra hacer que funcione en el navegador, debería funcionar sin problemas en cualquiera de estas cinco plataformas, debido a su gran compatibilidad.

En cambio, Android e IOS no pueden ofrecer eso, al menos no por ahora, porque uno usa un estándar propio y cerrado además de una política estricta y privativa para crear aplicaciones (IOS) y, el otro, usa como principal medio de desarrollo un estándar más "antiguo" aunque conocido, Java.

Por otra parte, una segunda motivación es que estos recientes sistemas operativos móviles no se enseñan a programar en la carrera y, aunque actualmente no son de los más conocidos, el hecho de que ahora sean de libre acceso hace que sean plataformas muy atractivas y con perspectivas de futuro.

1.3. Objetivos

Este proyecto se centrará en estudiar los sistemas operativos basados en Web en plataformas móviles mediante el desarrollo de varias aplicaciones con soporte multimedia, siendo este estudio el **objetivo principal** que se persigue en este proyecto.

Pero para poder alcanzar este objetivo se establecen los siguientes **objetivos específicos**:

- Comparar el desarrollo web frente al desarrollo con otros lenguajes (o nativos).
- Identificar los distintos sistemas operativos (basados en tecnologías web) disponibles y posibilidades de desarrollo que ofrecen en cuanto a entorno de programación.
- Estudiar y comprender cómo es la arquitectura y el funcionamiento de los sistemas operativos basados en Web para dispositivos móviles.
- Estudiar y manejar los entornos de desarrollo de las plataformas que van a ser evaluadas mediante pequeñas aplicaciones.
- Crear dos tipos de aplicaciones en *Firefox OS*, una que tenga acceso al *hardware* del dispositivo y otra que no.
- Realizar diversas pruebas con ellas en otros entornos y dispositivos, para posteriormente evaluar los resultados.

1.4. Fases de desarrollo

Para el alcance del presente proyecto, se decide dividir éste en diferentes fases, las cuales se indican a continuación en orden progresivo:

- **FASE 1: Documentación**

  Para empezar, se procedió a la búsqueda de información de estos entornos móviles para tener una mayor documentación y conocimiento al respecto.

- **FASE 2: Despliegue de los entornos de desarrollo**

  Esta fase involucra, en primer lugar, la instalación del sistema operativo Linux (Ubuntu 12.04 LTS), ya que debido a que *Ubuntu Touch OS* y *WebOS* se basan en Linux, se procede a instalarlo. Una vez se dispone de él, se ha de tener una máquina virtual (VirtualBox) a través de la cual se ejecutarán éstos. Para ello, a partir de un Terminal que ofrece Linux (Ubuntu, en este caso, ya que los SDKs se establecen mejor en este sistema que en Debian) se procede a ejecutarlos.

  Cabe destacar, que los otros tres sistemas operativos móviles serán ejecutados en Windows debido a que se tiene más experiencia y agilidad en este entorno. Por lo que para el desarrollo de las otras dos aplicaciones creadas en Fiferox OS se empleará dicho sistema operativo.

  En segundo lugar, puesto que una de las finalidades del proyecto es el estudio de los sistemas operativos *WebOS*, *Firefox OS*, *Tizen OS*, *Ubuntu Touch OS* y *Sailfish OS*, se comenzó por investigar los requerimientos necesarios para comenzar a crear aplicaciones con ellos. Para ello se comenzó por la instalación de los SDKs.

  Estos sistemas operativos móviles ponen a disposición de los programadores dos tipos de entorno de desarrollo, permitiendo, el primero, ejecutar las aplicaciones en desarrollo sobre un simulador o emulador y, el segundo, destinado para aquellos desarrolladores que deseen ejecutar las aplicaciones que hayan creado sobre el dispositivo físico. Este estudio se ha basado en ambos entornos, por lo que, en primer lugar, se procede a la instalación de dichos emuladores para aprender su funcionamiento y emplear las herramientas de desarrollo que contienen. Y, posteriormente, como se disponía de dispositivos con sistemas operativos Firefox y Android, se probaron las aplicaciones en ellos para comprobar su correcto funcionamiento.

  Por lo que esta fase se basa en la preparación del entorno, instalación de los distintos

SDKs y familiarización de las APIs.

- **FASE 3: Desarrollo de la primera aplicación en todos los entornos**

  Una de las aplicaciones que se proponían para comenzar a programar en estos nuevos entornos es el famoso "HolaMundo". Cabe destacar que se ha creado esta aplicación, como primera toma de contacto con cada uno de los sistemas operativos, con el fin de poder aportar más conocimientos para este estudio. Más adelante, en el Anexo B.1 y posteriores, se explicará con detenimiento los pasos que se han seguido para construir esta primera aplicación en cada entorno.

- **FASE 4: Desarrollo de las aplicaciones en Firefox OS**

  Adquiridos todos los conocimientos necesarios, se comienza con el estudio y la creación de la aplicación *ColorLight* en *Firefox OS*, (dado que se disponía de dispositivo) y, posteriormente, se continúa en este mismo entorno con *Images ScanApp*. Se desarrollan estas dos aplicaciones, debido a que la primera se decide que no tenga acceso al hardware del dispositivo para que no emplee APIs nativas y, la segunda, que hace uso de las APIs multimedia de *Firefox OS*, ya que ésta sí requiere uso de *hardware*. Cabe destacar que durante esta fase se realizaron consultas a los expertos a través de los diferentes foros de las páginas de desarrolladores.

- **FASE 5: Estudio y Evaluación del estudio**

  Una vez se han creado estas aplicaciones, se realizan una serie de pruebas (tanto en los emuladores como en los dispositivos reales) sobre el entorno que más se asemeja a *Firefox OS* en cuanto a características se refiere, así como en aquellos entornos cuyas características difieran de alguna manera. Finalmente, una vez se han obtenido las pruebas y se han evaluado, se indican las conclusiones obtenidas.

- **FASE 6: Redacción de la memoria**

  Con el fin de que el estudio sea lo más llevadero posible, se ha decidido ir elaborando y redactando la memoria a la vez que se crean las aplicaciones.

1.5. Medios empleados

A continuación se realiza una enumeración de los medios que han sido necesarios emplear para la realización de este proyecto:

Hardware:

- Dispositivo ZTE Open (Firefox OS)
- Dispositivo THL W8s (Android)
- Ordenador portátil HP Pavilion dv5
- Ordenador de mesa Packard Bell Imedia D5000

Software:

- S.O. Linux Ubuntu 12.04
- S.O. Windows 7 Ultimate
- Virtual Box v4.2 para Linux
- SDK WebOS para Linux (Ubuntu)
- SDK Ubuntu Touch para Linux (Ubuntu)
- SDK Tizen para Windows
- SDK Sailfish OS para Windows
- Eclipse IDE para desarrolladores JavaScript Web (Ubuntu)
- Tizen IDE (Windows)
- Qt Creator para Linux (Ubuntu)
- Sailfish IDE (Windows)
- Adobe Dreamweaver CS6 para Windows

- Adobe Photoshop CS6 para Windows
- TeXnicCenter para Windows
- Adobe Reader XI para Windows
- Microsoft Project Professional 2007

Otros:

- Conexión a Internet (Descarga de herramientas de desarrollo, documentación, etc.)

1.6. Estructura de la memoria

El desarrollo de esta memoria comenzará por el *Planteamiento del Problema*, en el que se hará un *Análisis del Estado del Arte* abordando el estudio sobre los distintos sistemas operativos móviles basados en Web, así como los lenguajes y APIs que emplean; y después, se indicarán los *Requisitos* del proyecto así como las *Restricciones y el Marco Regulador*. Posteriormente, se planteará el *Diseño de la Solución Técnica*, donde se instalarán los SDKs de los diferentes entornos, y se detallarán aquellas APIs que vayan a ser empleadas en el capítulo posterior. Tras lo cual, en dicho capítulo, denominado *Resultados y Evaluación* se expondrán las aplicaciones creadas, además de las conclusiones que se han obtenido en las pruebas que han sido realizadas durante todo el estudio.

La memoria prosigue con la presentación de la *Planificación y del Presupuesto* de este proyecto. Seguidamente, se exponen una serie de indicaciones para ampliar el estudio, mediante la funcionalidad de las aplicaciones en las que se ha llevado a cabo su desarrollo así como de algunos trabajos futuros que se podrían realizar. Y, finalmente, para concluir la memoria, se indicarán las *Conclusiones* obtenidas tras la realización del presente Trabajo Fin de Grado. Ambas partes se expondrán en *Futuras Líneas y Conclusiones*.

Capítulo 2

Planteamiento del problema

2.1. Análisis del Estado del Arte

Las nuevas tecnologías han propiciado el auge del uso de Internet en los dispositivos tales como teléfonos móviles y *tablets*. La Web se ha consolidado como el medio de más alto crecimiento en la historia, imprescindible hoy día para extraer todo tipo de información. Con esta culminación han surgido nuevas aplicaciones, donde *HTML 5*, *CSS 3* y *JavaScript* tienen especial protagonismo. Pero antes de realizar un análisis de éstos y de aquellas plataformas que se basan en el estándar *HTML 5*, se hará una comparación entre las aplicaciones nativas, web e híbridas.

2.1.1. ¿Aplicación nativa, web móvil o híbrida?

La batalla en el desarrollo móvil [2] va más allá de la lucha entre sistemas móviles como pueden ser iOS, Android o Windows Phone 7. *HTML 5* lleva tiempo generando altas expectativas y plantea el futuro del desarrollo móvil [3] entre las aplicaciones nativas y las web móvil que muchos de los desarrolladores acostumbrados a la Web tradicional pueden aprovechar [4].

Aplicaciones nativas

Las aplicaciones nativas son consideradas aplicaciones convencionales móviles, en otras palabras, están implementadas en el lenguaje nativo del propio terminal, por lo que todos lo recursos de éste estarán accesibles para poder sacar el máximo partido a la aplicación. Éstas residen en el dispositivo, suelen desarrollarse con lenguajes de programación que requieren una compilación anterior y se instalan directamente en el *smartphone*, a través de un medio de distribución de aplicaciones. De este modo, se consigue una mayor difusión de la aplicación y es más accesible para los usuarios. Además, al terminar la instalación de la aplicación, el usuario dispondrá de un acceso directo de ésta para poder lanzarla de una forma rápida y sencilla. Cabe destacar también, que este tipo de aplicaciones reciben las notificaciones y las actualizaciones automáticas al instante, es decir, éstas pueden funcionar aunque el teléfono no esté conectado a Internet, así los usuarios pueden acceder a la información de la aplicación en cualquier momento. Sin embargo, para que la aplicación esté disponible para todos los mercados, debe ser programada para cada plataforma.

La gran desventaja [5] de una aplicación nativa es que si se quiere lograr una aplicación profesional, se necesita un nivel de conocimiento de programación y una experiencia avanzados, lo que hace que la implementación sea más costosa. Otro inconveniente, como se comentó anteriormente, sería que las aplicaciones nativas necesitan un desarrollo para cada uno de los sistemas operativos e incluso para versiones diferentes de éstos, ya que son programas desarrollados específicamente para una plataforma y sus interfaces siguen los estándares y normas de dicho entorno, incrementándose el tiempo de desarrollo, lo que supone aún más coste. Además, el usuario deberá de actualizar manualmente la aplicación a través de los diferentes *market places*, al contrario que las aplicaciones web que siempre que el usuario acceda a ellas estarán actualizadas directamente sin que éste haga nada. Asimismo, para los desarrolladores también hay una pega, y es que a la hora de publicar la aplicación, éstos se enfrentarán a los procesos de validación de los diferentes medios de distribución de aplicaciones, siendo algunos más exigentes que otros.

Aplicaciones web

A día de hoy, los *smartphones* vienen con potentes navegadores que soportan múltiples funciones: *HTML 5* [1], *CSS 3* [2] y *JavaScript* [3]. Los proveedores de aplicaciones pueden crear páginas web que reconocen el acceso desde el móvil y actúan de manera que se pueda ver en una pantalla más pequeña. También se puede diseñar una aplicación web que sea muy similar a una nativa, aunque realmente no lo sea.

Para este tipo de aplicaciones el problema que se presenta es que los lenguajes web no tienen todavía compatibilidad con todas las funciones nativas (API) [4], tales como el GPS, acelerómetro, cámara, agenda de contactos, calendario, etc.

En cambio, una habilidad que se destaca sobre las aplicaciones web en comparación con las nativas, es su gran versatilidad, debido a que son multiplataforma, lo que significa que se puede ejecutar en cualquier sistema operativo, basta con que el terminal disponga de un navegador. Esto conlleva a que no es necesario tener que volver a programar, por lo tanto, esto disminuye los problemas que traen las aplicaciones nativas creadas para un único sistema operativo que requieren atención constante, lo que implica un aumento de los costes.

Asimismo, si comparamos éstas con las nativas, las aplicaciones web requieren un tiempo invertido relativamente corto, debido a que la programación en *HTML 5* es mucho más simple y esto hace que se necesiten menos requisitos. Y las ventajas siguen en aumento, ya que en caso de actualización, no es necesario que el usuario deba actualizar la aplicación, pues al ser una página web, siempre accederá a la versión más actualizada y, además, al no ser descargable desde los medios de distribución de aplicaciones, no es necesario pasar ningún proceso de validación.

[1] *HTML 5:* Lenguaje para estructurar y presentar el contenido de la www (*world wide web*).

[2] *CSS 3:* Utilizado para controlar el estilo y el diseño de páginas web.

[3] *JavaScript:* Lenguaje de programación avanzado, bien conocido por su uso en la construcción de dichas páginas.

[4] *API:* Interfaz de Programación de Aplicaciones.

Un punto negativo para las aplicaciones web es la promoción, ya que su presencia en los mercados oficiales de los sistemas operativos predominantes como iOS y Android no está admitida. Esto hace que el descubrimiento de las aplicaciones sea más complicado, lo que supone un esfuerzo mayor en el sector de *marketing*. Otra desventaja es el rendimiento de las aplicaciones web, que sigue siendo mucho menor porque éstas son mucho más lentas en comparación con las aplicaciones nativas. Y, por último, cabe destacar que para algunos tipos de aplicaciones será necesario contar con una conexión a Internet para que funcionen.

Aplicaciones híbridas

Este tipo de aplicación se utiliza cuando se quiere desarrollar una aplicación móvil muy flexible que combina elementos web y nativos. Desde el punto de vista técnico, una aplicación híbrida es una aplicación nativa construida con *HTML 5*.

La aplicación híbrida contará con todos los beneficios que le proporciona una aplicación nativa, como por ejemplo, el acceso a todas las APIs, ya que algunas partes de la aplicación serán desarrolladas utilizando tecnologías web. El problema surge debido a que empresas como Apple no aceptan aplicaciones que intentan infiltrar contenido que no puede ser revisado por ellos mismos. En definitiva, este tipo de aplicaciones adquieren cada una de las ventajas e inconvenientes de las aplicaciones anteriores.

Una vez se han comentado las diferencias, se procede a explicar de manera más detallada los distintos lenguajes que emplean las aplicaciones web móvil.

2.1.2. Histórico

W3C (*World Wide Web Consortium*) fue creada en 1994 como un consorcio, para que los miembros de la industria pudieran unirse para establecer la normativa juntos, y de ese modo, evitar las incoherencias en las implementaciones individuales.

HTML 4 fue lanzado en diciembre de 1997, pero en la década del 2000 las normas parecía que ya no evolucionaban. Insatisfechos, un grupo dentro de la compañía de Apple,

la Fundación Mozilla y Opera Software crearon WHATWG (*Web Hypertext Application Technology Working Group*), para continuar con el desarrollo del estándar *HTML* y con el conjunto de APIs (*Application Programming Interface*) para las aplicaciones web.

El resultado que hoy conocemos como *HTML 5*, es el equivalente a un conjunto de tecnologías que constituyen la Web moderna, permitiendo animaciones, multimedia y aplicaciones más sofisticadas gracias a *CSS 3* y las nuevas capacidades de *JavaScript*. De hecho, el uso fuerte de *JavaScript* ha ayudado a mejorar esto gracias a *frameworks* como *jQuery*, *jQuery UI*, *Sproutcore*, entre otros. Dicho estándar fue publicado por W3C en 2008 y se espera una publicación más avanzada a lo largo de este año, 2014. *WHATWG* sigue trabajando para *HTML*, por lo tanto, las especificaciones para el mismo se consideran como un "estándar vivo", siendo *HTML 5* solo un resultado de su evolución.

2.1.3. HTML 5 - HyperText Markup Language

Desde que apareció *CSS*, el papel de *HTML* se ha limitado a la definición de la estructura de un documento, no a su apariencia visual. *HTML 5* [6] pone aún más énfasis en la semántica, introduciendo nuevos elementos que permiten a los autores definir con más precisión algunas partes del documento. De esta manera, *HTML 5* [7] será mejor comprendido y tratado que los demás [8].

Esta nueva versión del lenguaje básico de la Web [9], proporciona mecanismos para simplificar el trabajo y facilitar la inclusión de elementos multimedia. El principal criterio de diseño de *HTML 5* ha sido el de resolver problemas prácticos, y con este objetivo adopta soluciones dirigidas a facilitar el trabajo en situaciones reales. Además, incluye elementos nuevos destinados a enriquecer la presentación de documentos. Son ejemplos de ello los elementos semánticos *article, header, hgroup, nav, section, aside* y *footer*. Con ellos se pretende evitar que los autores abusen del elemento *div* para delimitar partes de un documento.

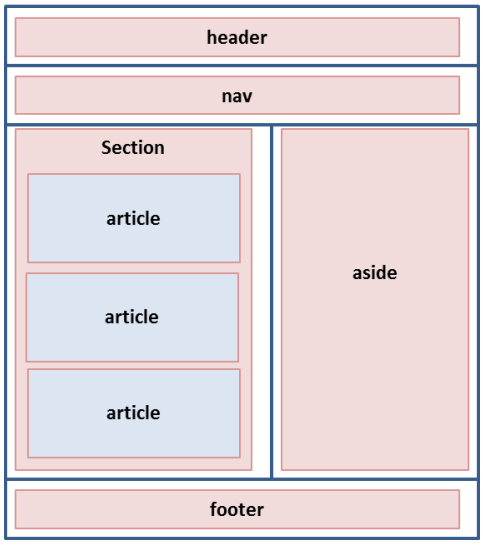


Figura 2.1: Ejemplo de la estructura de un página en HTML 5 - Fuente: www.julitogtu.com

Por otra parte, existe un medio para que el usuario pueda hacer llegar datos de entrada a un servidor: los formularios, que recogen información que después remiten a aplicaciones que se ejecutan en el servidor. *HTML 5* define más de una docena de nuevos controles (*email, range, date, time, placeholder, autofocus, etc.*) que actúan, por fin, sin necesidad de utilizar *JavaScript*, un lenguaje de programación que no todos los usuarios tienen habilitado.

Este nuevo estándar propone que sean los navegadores, y no los creadores de contenido, quienes faciliten la entrada y la validación de datos que tienen un patrón regular o están sometidos a restricciones. Así, el dolor de cabeza que supone para los diseñadores de páginas web el verificar el formato de direcciones electrónicas, intervalos de valores, términos de búsqueda, colores, fechas y horas, entre otros tipos de datos, se descarga ahora sobre los navegadores.

Para incrustar contenido multimedia, *HTML* ya contaba con el elemento *object*, pero la nueva versión del estándar hace una propuesta más semántica. En el ámbito multimedia, *HTML 5* incorpora directivas nuevas que actúan como contenedores de vídeo, gráficos vectoriales y audio.

2.1.4. CSS 3 - Cascading Style Sheets

CSS es un lenguaje que define la forma de salida para los documentos de tipo *HTML*. *CSS* incluye colores, tipos de letra, imágenes, vídeos, líneas, altura, anchura, imágenes de fondo y muchas otras más opciones. *HTML* muchas veces está mal utilizado para crear el diseño de las páginas web, ya que *CSS* ofrece más opciones, es más preciso y sofisticado. Además, es compatible con todos los navegadores actuales.

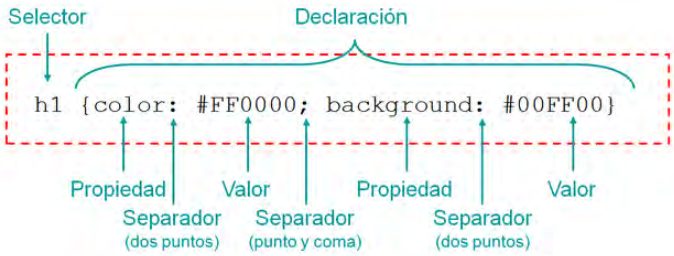


Figura 2.2: Elementos de CSS 3 - Fuente: www.desarrolloweb.dlsi.ua.es

Cabe destacar, que este lenguaje se puede definir tanto en la cabecera de una página web, como en un archivo separado con respecto al documento de *HTML*. La mejor opción es esta última, pues separando los archivos *CSS* que definen el diseño de la página en sí, facilita enormemente su mantenimiento. Esta separación puede mejorar la accesibilidad de los contenidos, proporcionar más flexibilidad y reducir la carga de mantenimiento de una página web al ofrecer un control sencillo. Este tipo de archivo, también puede reducir la complejidad y la repetición de etiquetas que se utilizan para dar formato a la estructura de contenidos. Como se puede apreciar todo lo que ofrece este lenguaje son ventajas.

2.1.5. JavaScript

JavaScript [10] es un lenguaje utilizado generalmente para la programación del lado del cliente, *client-side*. El navegador descarga el código de *JavaScript*, que se ejecuta e interpreta en el ordenador, no en el servidor. Aunque la programación *client-side* es el uso más popular de *JavaScript*, más reciente se puede programar con *JavaScript server-side*. Se trata de un lenguaje de programación simple, diseñado para añadir interactividad a las páginas *HTML* en las que, normalmente, está incluido; como por ejemplo, puede poner

texto dinámico en las páginas *HTML*, reaccionar a los eventos, leer y escribir elementos en *HTML*, detectar el navegador de usuario. Además, se puede utilizar para validar los datos de un formulario antes de ser enviado al servidor así como para crear *cookies*. Por lo que tiene multitud de utilidades como se puede ver.

Figura 2.3: Fragmento de código JavaScript

Al igual que en *CSS*, éste se puede definir en un fichero externo o incluirlo en la página (dentro del *HTML*). Se recomienda emplear la primera opción y separar todo según el tipo que sea para que se puedan identificar mejor las características de cada uno de ellos. De este modo, el código queda más limpio e inteligible.

- *Java y JavaScript*

  Java y *JavaScript* son dos lenguajes completamente distintos tanto en diseño como en concepto, es más que nada un parecido de los nombres. *Java* (desarrollado por *Sun Microsystems*) es un lenguaje más potente y complejo, al igual que *C* y *C++*. En cambio, *JavaScript* fue desarrollado por primera vez por *Netscape*, con el nombre de *Live Script*, que era un lenguaje de programación que extiende las capacidades de *HTML*.

- *Frameworks con compatibilidad HTML 5 y CSS 3*

  Un *framework* [11] está diseñado para apoyar el desarrollo de sitios web dinámi-

cos, aplicaciones web y servicios web. Este tipo de *frameworks* intentan aliviar el exceso de carga asociado con actividades comunes usadas en desarrollos web. A continuación, se muestran unas recomendaciones de algunos de ellos con compatibilidad *HTML 5* y *CSS 3* [12].

1. *Boilerplate*

   Facilita el proceso de creación de sitios web, además de ofrecer compatibilidad con prácticamente todos los navegadores y un diseño específico para móviles. Utiliza las últimas versiones de *jQuery* y de *Modernizr*.

2. *960 Grid*

   Es un sistema de rejilla para hacer páginas con un ancho de 960 píxeles. Tiene dos variantes de 12 y 16 columnas. A partir de esa estructura de columnas va a permitir configurar cualquier combinación de tamaños y disposiciones en el diseño de la Web.

3. *Gridless*

   Destinado para la creación de sitios web para móviles. Permite utilizar la normalización *CSS*, una cuidada tipografía y una buena estructura de carpetas, ofreciendo una gran compatibilidad con prácticamente cualquier navegador.

4. *Baseline*

   Facilita el diseño con una rejilla muy bien conseguida y unas tipografías muy agradables. Este *framework* incluye estilos para elementos de formulario y es compatible por la mayoría de los navegadores (*Safari 3+, Google Chrome, Firefox 3+, Opera 9+ e IExplorer 9*).

5. *Sproutcore*

   Se trata de un *framework* de código abierto para la construcción de aplicaciones web que necesitan máxima velocidad, además de proporcionar innovadoras experiencias de usuario en la Web, permite crear aplicaciones a nivel escritorio para los navegadores más actualizados ya que no depende de ningún plugin externo.

- *jQuery* [13] [14]

  jQuery es una biblioteca de *JavaScript*, que permite simplificar la manera de interactuar con los documentos *HTML*, manipular el árbol *DOM*[5], manejar eventos, desarrollar animaciones y agregar interacción con la técnica *AJAX*[6] a páginas web. En estos momentos, *jQuery* es la biblioteca de *JavaScript* más utilizada [15].

  jQuery es un software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privativos. *jQuery*, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en *JavaScript* que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio [16].

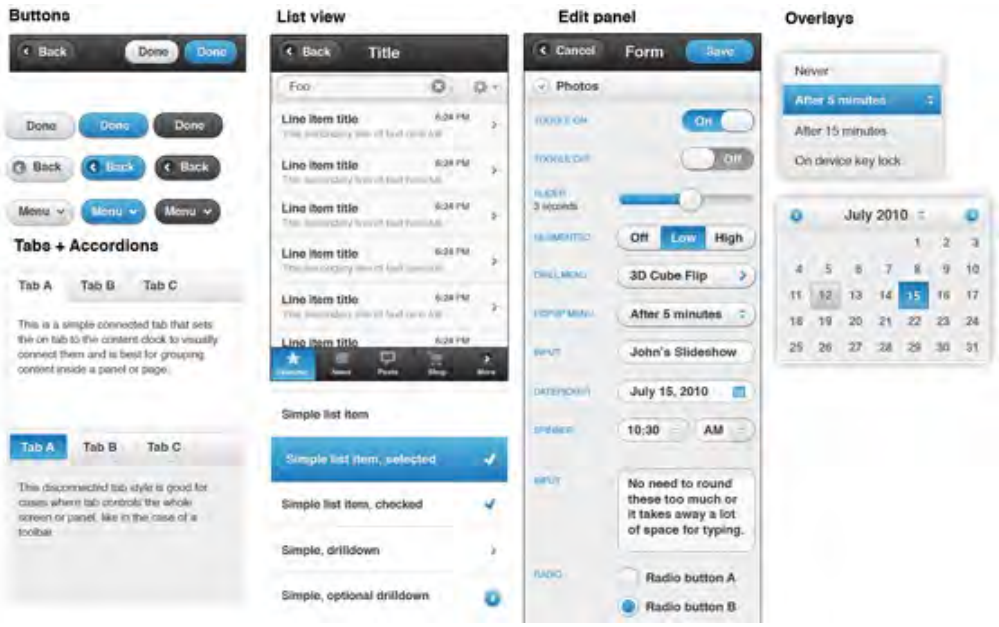


Figura 2.4: Funcionalidades del framework jQuery Mobile - Fuente: www.pabloaraya.org

[5] *DOM:* Modelo de Objetos del Documento es una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos *HTML* y *XML*, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

[6] *AJAX:* Acrónimo de *Asynchronous JavaScript And XML*, es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano.

2.1.6. Sistemas Operativos que emplean Aplicaciones Web

El mercado de la telefonía móvil es uno de los mercados más dinámicos dentro de la electrónica de consumo. Al contrario de lo que ocurre con televisores o las minicadenas, el ciclo de cambio de producto es bastante más corto y la evolución de las tecnologías asociadas es mucho más acelerada. A consecuencia de esto, se han producido bastantes cambios en el panorama de los sistemas operativos móviles en los últimos años y se intuyen más por venir en el próximo. De hecho, muy recientemente han llegado nuevos sistemas operativos móviles con sus correspondientes filosofías de productos y apoyos tecnológicos. A continuación se exponen cada uno de éstos:

- *Firefox OS mobile* [17]

  Quizá el sistema de Mozilla es el mejor posicionado de todos [18], ya que cuenta con un gran apoyo de fabricantes y operadores. Además, Firefox tiene una gran comunidad de desarrolladores detrás, que ven con buenos ojos la llegada de este sistema operativo. De hecho, la fundación Mozilla también está realizando grandes esfuerzos para potenciar *Firefox OS*. Se une también la combinación de apoyo de fabricantes y operadores, la comunidad de desarrolladores y las propias características del sistema, convirtiendo a éste en uno de los favoritos para hacerse con algo de cuota de mercado.

  Firefox OS es un sistema operativo móvil, basado en Linux, de código abierto, creado con el fin de emplearse en *smartphones* y tabletas. Este sistema operativo está enfocado especialmente en los dispositivos móviles, incluidos los de gama baja. Está diseñado para permitir a las aplicaciones *HTML 5* comunicarse directamente con el *hardware* del dispositivo usando *JavaScript* y *open web API*. Al estar basado en *HTML 5*, facilita el desarrollo de aplicaciones que sirven tanto para la Web como para equipos móviles, como en este caso, consumiendo menos recursos, que afectará de manera positiva en la duración de la batería. Aunque el hecho de que se base en el propio navegador podría hacer pensar a priori que las aplicaciones serían sobre todo herramientas *online* -actualmente muchas lo son-, pero también hay soluciones

que se instalan en local y se pueden usar sin conexión a Internet.

Cabe destacar que *Firefox OS* ofrece algunas utilidades básicas y muy populares (*Facebook* y *Twitter* son las claras destacadas), existen otras muchas a las que no será posible acceder. Por ejemplo, por el momento no hay cliente de *WhatsApp* (aunque se está implementando), ni tampoco soluciones destacadas en el ámbito de los juegos. Sin embargo, *Firefox OS* es una plataforma que demuestra su apertura también en este aspecto. Por ejemplo, en el caso de la mensajería instantánea han surgido proyectos como *Loqui*, un desarrollo *Open Source* que ya está disponible en versión *pre-alpha* para *Firefox OS* y que con su soporte del protocolo *Jabber* y de cuentas de *Google Talk* solventa en parte esas necesidades.

Las ventajas de *Firefox OS* son claras. Es una plataforma totalmente abierta y sin APIs propietarias. Esas ventajas parecen superar a sus desventajas, entre las que destacaríamos su reducido catálogo de aplicaciones en comparación con otros sistemas operativos móviles, o la incertidumbre sobre la capacidad real de las aplicaciones y juegos *HTML 5*.

En cuanto a la arquitectura de *Firefox OS* [19], existen tres componentes [20] de gran relevancia:

1. *Gonk*

   Es el "sistema operativo" de bajo nivel de *B2G*[7]. A grandes rasgos, consiste en un *kernel* Linux y una capa de abstracción de *hardware*.

2. *Gecko*

   Es el entorno de ejecución. En *Gecko* están implementados los estándares de *HTML*, *CSS* y *JavaScript* y permite que esas interfaces se ejecuten correctamente en los distintos sistemas operativos. Esto significa que *Gecko* consiste

[7] *B2G: Boot to Gecko* tiene como objetivo crear un sistema operativo completo y autónomo para la Web abierta.

en una serie de pilas de gráficos, un motor de dibujado y una máquina virtual para *JavaScript*, entre otras cosas.

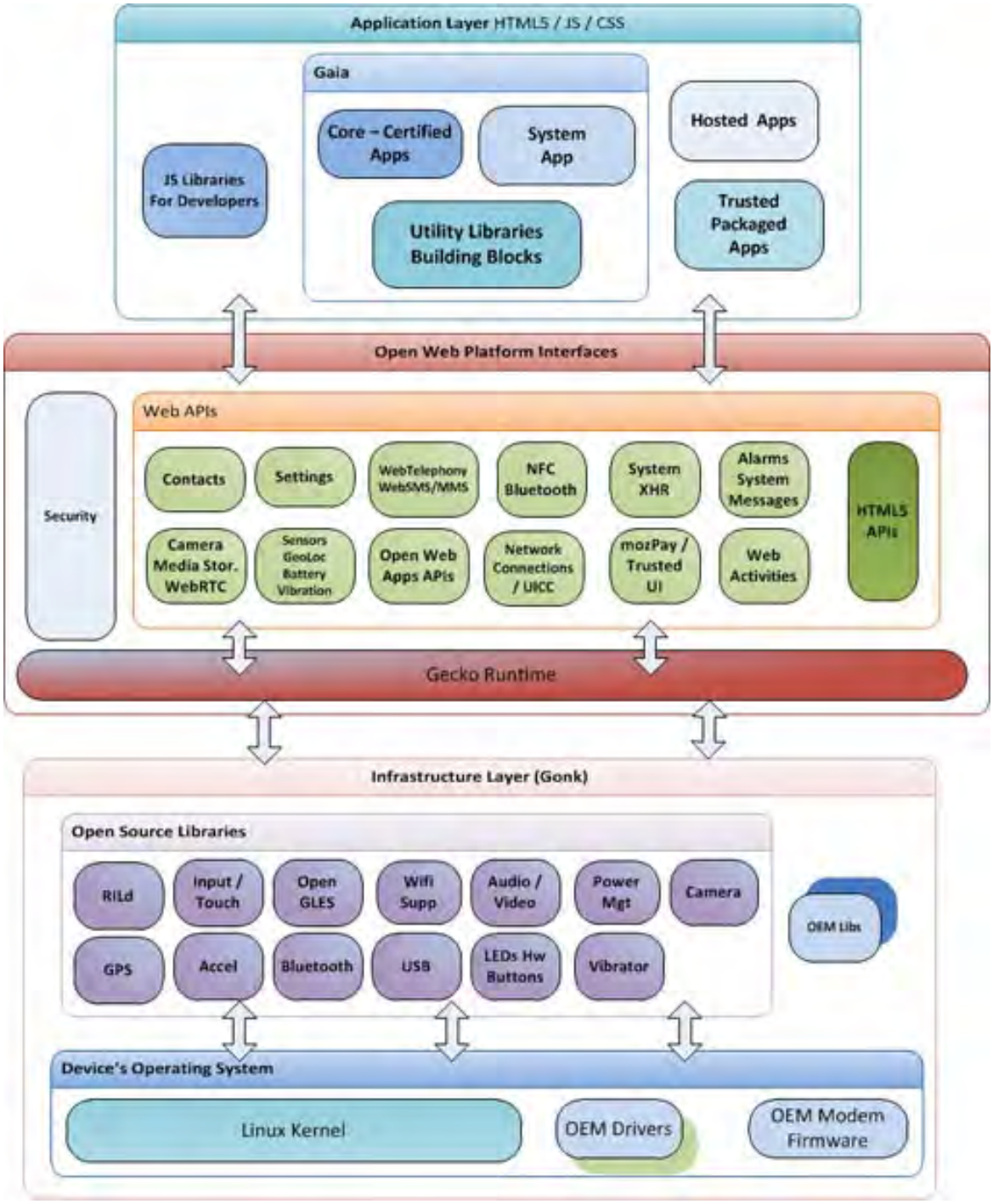


Figura 2.5: Arquitectura Sistema Operativo Firefox - Fuente: www.paradigmatecnologico.com

3. *Gaia*

   Es la interfaz gráfica del sistema operativo. Todo lo que aparece en la pantalla desde que *B2G* se inicia, es parte de Gaia. Es decir, las aplicaciones tales como la pantalla de bloqueo, el marcador telefónico, la aplicación de mensajes de texto, etc, son parte de Gaia. Esta interfaz gráfica está escrita enteramente en *HTML*, *CSS* y *JavaScript*.

Uno de los principales objetivos de Mozilla fue diseñar, especificar e implementar una serie de funciones así como sus correspondientes interfaces con el fin de crear un conjunto de APIs [21] (*WebAPI*) y ofrecer un amplio abanico de utilidades que hacen que se pueda acceder de una manera más sencilla a las funcionalidades específicas de los dispositivos como la cámara, bluetooth, telefonía, localización, red, etc.

A continuación se muestra una tabla en la que se pueden ver algunas de las APIs de las que componen la *WebAPI*:

Tabla 2.1: *APIs que conforman la WebAPI*

Communication APIs	*Network Information API, Bluetooth, Mobile Connection API, Network Stats API, TCP Socket API, Telephony, WebSMS,...*
Hardware access APIs	*Ambient Light Sensor API, Battery Status API, Geolocation API, Pointer Lock API, Proximity API, Device Orientation API,...*
Data management APIs	*FileHandle API, IndexedDB, Contacts API, Device Storage API, Settings API*
Other APIs	*Alarm API, Simple Push API, Web Notifications, Apps API, Web Activities, WebPayment API, Browser API,...*

Además, en la página Wiki *WebAPI* [8], se puede obtener más información sobre estas nuevas APIs. En dicha web se puede encontrar una lista actualizada de todas las APIs, su estado de implementación en las diferentes plataformas de Firefox, los organismos de normalización involucrados, etc. como se puede ver a continuación:

[8] *Wiki WebAPI:* https://wiki.mozilla.org/WebAPI

Planned for initial release of B2G (aka Basecamp)

Unless otherwise noted, APIs listed in the table below were available in Firefox OS v1.0.1 or will be available in v1.1.

API	Description	Standardization	Availability		
WebTelephony	Allow placing and answering phone calls as well as build in-call UI.	W3C ED (SysApps)	D	A	B
Vibration API	Control device vibration for things like haptic feedback in games. Not intended to solve things like vibration for notification.	W3C CR (Device APIs)	D	A	B
WebSMS	Send/receive SMS messages as well as manage messages stored on device.	W3C ED (SysApps)	D	A	B
Idle API	Get notifications when user is idle.	Needs plan	D	A	B
Screen Orientation	Get notification when screen orientation changes as well as control which screen orientation a page/app wants.	W3C WD (WebApps)	D	A	B
Settings API	Set system-wide configurations that are saved permanently on the device.	Future?	D	A	B
Power Management API	Turn on/off screen, cpu, device power, etc. Listen and inspect resource lock events.	Future?	D	A	B
Mobile Connection API	Expose signal strength, operator, etc for GSM and other mobile connections. This does not cover WiFi.	Future?	D	A	B
TCP Socket API	Low-level TCP socket API. Will also include SSL support.	W3C ED (SysApps)	D	A	B
Geolocation API	Access to the end user's location.	W3C CR	D	A	B
WiFi Information API	Privileged API to get a list of available WiFi networks. Also get signal strength and name of currently connected network, etc.	Future?	D	A	B
Device Storage API	Add/Read/Modify files stored on a central location on the device. For example the "pictures" folder on modern desktop platforms or the photo storage in mobile devices.	Needs plan	D	A	B
Contacts API	Add/Read/Modify the device contacts address book.	W3C ED (SysApps)	D	A	B
Mouse Lock API	Lock access to mouse and get access to movement deltas rather than coordinates.	W3C ED	D	A	B
Open WebApps	Install web apps and manage installed webapps. Also allows an installed webapp to get payment information. Everything needed to build a Open WebApps app store.	W3C WD (SysApps)	D	A	B
WebBluetooth	Low level access to Bluetooth hardware.	Future?	D	A	B
Network Information API	Get basic information about current network connectivity. Example: "How fast of a connection do I have?".	W3C ED	D	A	B
Battery Status API	Information about battery charge level and if device is plugged in.	W3C CR (DAP)	D	A	B
Alarm API	Schedule a notification, or for an application to be started, at a specific time.	W3C WD (SysApps)	D	A	B
Browser API	Enables implementing a browser completely in web technologies.	Future?	D	A	B
Time/Clock API	Set current time. Timezone will go in the Settings API.	Future?	D	A	B
Web Activities	Delegate an activity to another application.	Discussed in Device APIs	D	A	B
Push Notifications API	Allow the platform to send notification messages to specific applications.	W3C ED (Webapps)	D	A	B
Permissions API	Allow Settings app to manage all app permissions in a centralized location	Future?	D	A	B
WebFM API	For FM radio feature.	Future?	D	A	B
FileHandle API	Writable files with locking.	Needs plan	D	A	B
Network Stats API	Monitor data usage and expose data to privileged apps	Future?	D	A	B
WebPayment	Allow web content to initiate payments and refunds for virtual goods. For the server implementation, see WebPaymentProvider.	Beginning	D	A	B
IndexedDB	Client-side storage of structured data and high performance searches on this data	W3C WD	D	A	B
Archive API	Blob support for Zip file contents	Future?	D	A	B
Ambient light sensor	Device light sensor support	W3C WD	D	A	B
Proximity sensor	Device proximity sensor support	W3C WD	D	A	B
SystemXHR	External (non-same origin) XMLHttpRequests	?	D	A	B

Figura 2.6: Listado de APIs planificadas para el lanzamiento inicial de B2G -Boot to Gecko- (Wiki WebAPI)

Planned for the future

API	Bugs	Description	Progress	Availability
Resource lock API	bug 697132	Prevent resources from being turned off, for example screen dimming, WiFi turning off, CPU going into sleep mode etc.	Complete. Security Design Complete	
UDP Datagram Socket API	bug 745283	Low-level UDP API.	Planning. (Not P1 for basecamp)	
USB file-reading API	bug 748350 bug 737153	When enabled, allows mounting of device storage as a USB filesystem on the tethered computer.	Must be complete by June/July. Not really a webAPI, no security design.	
Camera API		This is part of the larger WebRTC effort. This is a big piece of work so see the link.	API and implementation underway. Security Design Complete	
Peer to Peer API		This is part of the larger WebRTC effort. This is a big piece of work so see the link.	API and implementation underway.	
WebNFC	bug 674741	Low level access to NFC hardware. So far focusing on NDEF support.	API drafted, implementation underway for B2G (not Andorid/Desktop), must be complete by June/July. Security Design Complete	
WebUSB	bug 674718	Low level access to USB hardware.	Security Design Complete	
HTTP-cache API		Query what's stored in the browsers http-cache. Add/remove entries. Update expiration time. Get data directly from cache.	None	
Calendar API		Add/Read/Modify to the device calendar.	Implementation not planned ATM. If/when implemented, it should mimic WebAPI/ContactsAPI.	
Spellcheck API		Enable webpages to check if a piece of text is correctly spelled as well as get suggestions for corrections.	None	
Background services		Enable a web application to run in the background and perform tasks like syncing or respond to incoming messages.	Initial proposal of API. Security Design Active	
LogAPI		Allows to register the user activity on the phone.	API proposal exists. Not planned for 1.0.	
Keyboard/IME API	bug 737110 (WebAPI mailing list post, Extended API mailing list post)	Enables implementing virtual keyboards.	Only exposed to certified apps for V1. Controlled via a setting instead.]	D A B

Legend

D = Desktop, A = Android, B = B2G
only available to certified apps on this platform
only available to privileged and certified apps on this platform
implemented and preference enabled by default on on this platform
implemented but requires explicitly turning on the preference on this platform
not implemented for this platform
not currently planned for this platform

Figura 2.7: Listado de APIs planificadas para ser mejoradas en un futuro

Asimismo, cabe destacar que Firefox es compatible con el modo de pantalla completa nativo en OS X Lion 10.7 para una mejor experiencia con vídeos y juegos web. También es compatible con las comunicaciones web en tiempo real (WebRTC), haciendo así posibles funciones como las videollamadas y el uso compartido de archivos entre navegadores. Y, por último, incluye un visor de PDF integrado con el que se pueden leer documentos PDF directamente en el navegador, lo que facilita su lectura porque no hay que descargar el contenido ni recurrir a ningún plugin de lectura.

Además, cabe indicar que el acceso a las nuevas APIs está controlado por un sistema de permisos, existiendo tres tipos de aplicaciones: *web apps, privileged apps* y *certified apps*. Estos permisos se declaran en el manifiesto de la aplicación, deter-

minando éste la lista de permisos que se pueden solicitar.

- Aplicaciones web: La mayoría de las aplicaciones de terceros serán aplicaciones "web", que es el tipo predeterminado, y solicita permisos adicionales además de los que ya están expuestos a la Web. Estas aplicaciones se pueden instalar desde cualquier sitio web, sin necesidad de más.

- Aplicaciones privilegiadas: Estas aplicaciones están autorizadas para pedir un aumento de los permisos al usuario.

- Aplicaciones certificadas: Este tipo de aplicaciones actualmente sólo pueden estar preinstaladas en el dispositivo, requiere esta política de seguridad de contenido para ser alojada en los servidores de Mozilla. De esta manera pueden dar acceso al *hardware*, pero minimiza el potencial de abuso y el *malware* a la vez.

- *Tizen OS* [22]

  El proyecto de Samsung, Intel y Panasonic va por muy buen camino gracias a su universalidad, la gran ventaja de poder usar aplicaciones de Android, tener una base en *HTML 5* y *Javascript* basado en la plataforma *WebKit*, -siendo una programación que le permite implementarse tanto en dispositivos móviles como en ordenadores de función y también en electrodomésticos-, hacen que este sistema operativo móvil tome relevancia. De hecho ya está liberado el código fuente y el SDK de Tizen 1.0, cuyo nombre en clave es *Larkspur* y, además, la versión deja de ser Beta. Este SDK incorpora varias mejoras: un nuevo simulador web para desarrolladores, el código fuente mejora el soporte de *HTML 5*, *WiFi Direct*, servicios de geolocalización y la interfaz, parecida en gran medida a *TouchWiz*.

  En cuanto a su arquitectura, Tizen está basado en Linux y su origen tuvo que ver con MeeGo, cuyo un sistema operativo era la unión de otros proyectos similares, como Moblin (propiedad de Intel y la *Linux Foundation*) y Maebo (un proyecto creado por Nokia). Además, como se comentó anteriormente es capaz de ejecutar aplicaciones Android, lo que le brinda acceso a miles de aplicaciones.

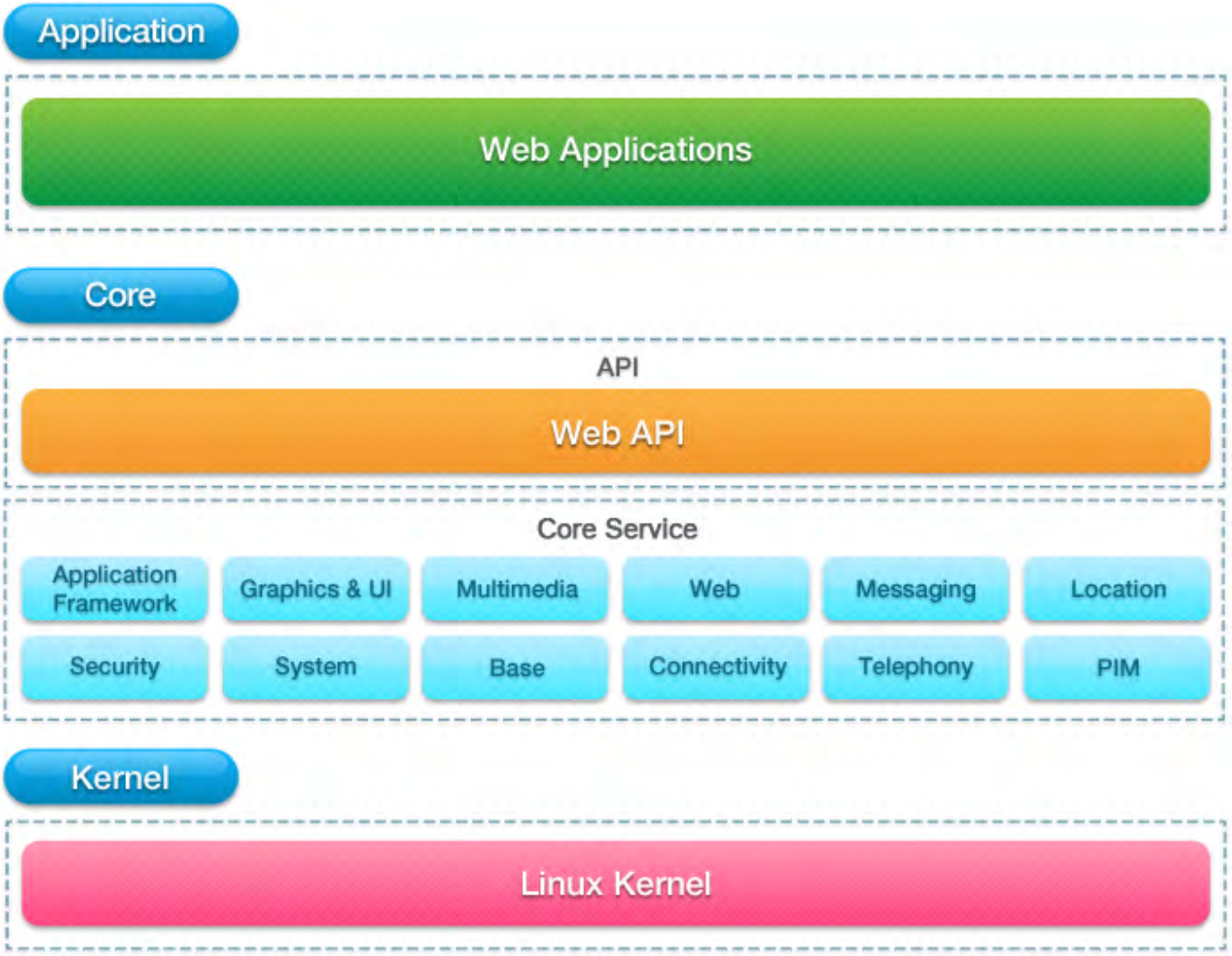


Figura 2.8: Arquitectura Sistema Operativo Tizen - Fuente: www.tizenspain.com

La ventaja que tiene *Tizen OS* con respecto a iOS y *Firefox OS* es que al ser compatible con aplicaciones de Android el ecosistema ya está creado, con lo cual muchos fabricantes podrían lanzarse a explorar.

Pero no todos son ventajas en este entorno, ya que aunque originalmente fue presentado como un sistema operativo de código abierto [23], Tizen 2 ha complicado su modelo de licencias. Su SDK está construido sobre componentes de código abierto, pero el SDK completo ha sido publicado bajo una licencia de Samsung de código no abierto.

Otro de los inconvenientes es que aunque Open Mobile, la empresa que permitirá correr aplicaciones de Android en Tizen, indicó que la compatibilidad se establecía en un 100 %, el emulador, finalmente, no se ofrecerá para que cualquiera pueda instalarlo, sino que la integración en *smartphones* y tabletas estará a cargo de las operadoras y distribuidores.

Por otro lado, Tizen también se decanta por crear un conjunto de APIs [24] por el mismo motivo que *Firefox OS*. En este caso, existen varias APIs y, dependiendo del tipo de aplicación creada (tipo web, nativa o híbrida) se podrán emplear unas u otras. A continuación se indica qué aporta cada una de ellas:

- API para el desarrollo basado en Web

Este API se divide en las siguientes:

1. **Web Device API:** Es el API básico de desarrollo para las aplicaciones basadas en Web, de ésta se obtiene la mayor parte de los elementos que se emplean en la programación más habitual.
2. **Web UI Framework API [25]:** Aquí se encuentran aquellos elementos que están relacionados con la programación a nivel gráfico, cómo es la creación de widgets, el control de eventos, los efectos y las animaciones.
3. **W3C/HTML5 API [26]:** Se compone de la especificación de *Tizen OS* para los estándares de la *W3C* y *HTML5*.

De las tres anteriores, probablemente la más importante es la primera, ya que es la que nos permite acceder a todas las capacidades del dispositivo. Desde este API, que está basado en *JavaScript*, se puede controlar todo el ciclo de vida de la aplicación, acceder a los datos del terminal, realizar intercambio de datos, etc.

En la tabla siguiente se muestra la división de las diferentes funcionalidades que proporciona este API:

Tizen Common	*Tizen*	Objeto base para el acceso a todo el API Web
Application	*Alarm*	Provee de funcionalidad para manipular las alarmas del terminal
	Application	Facilita información sobre las aplicaciones que se encuentran en ejecución o instaladas y da el control sobre ellas
Communication	*Bluetooth*	Acceso y control del Bluetooth
	Messaging	Controla todo lo que tiene que ver con el sistema de mensajería, tanto SMS, MMS, como el envío y recepción de emails
	NFC	Control para la realización de transacciones con NFC
Content	*Content*	Se emplea para acceder al contenido multimedia (imágenes, videos o música)
	Download	Proporciona interfaces y métodos para descargar objetos remotos vía peticiones HTTP
Input / Output	*Filesystem*	Para acceder al sistema de archivos del dispositivo. Se prevé que este API quede obsoleto en un futuro cuando se comience a utilizar el API de la W3C de acceso a ficheros
Social	*Calendar*	Control y gestión de la información del calendario
	Call History	Acceso al registro e histórico de llamadas y VoIP
	Contact	Todo lo referente a la administración de la información de los contactos del terminal
System	*Power*	Permite el control del uso de la batería
	System Information	Acceso al tipo de pantalla, características de la red, almacenamiento y resto de capacidades que tiene el dispositivo
	System Setting	Controla todos los ajustes (settings) del terminal
	Time	La usaremos para obtener información sobre fechas, horas y zonas horarias
User Interface	*Notification*	Se utiliza para notificar al usuario de cualquier tipo de evento que sucede dentro de la aplicación

Tabla 2.2: Funcionalidades Web Device API

- API para el desarrollo nativo

En este caso, se dispone de un conjunto de espacios de nombres (*namespaces*) en donde se especifican una serie de clases que se pueden invocar directamente en el código, muchos de estos *namespaces* comparten nombres muy similares a los que se han indicado en la tabla anterior. No obstante, hay que tener en cuenta que el desarrollo nativo y, por tanto, el acceso a estos espacios de nombres, siempre se realizará en *C++*.

A continuación se muestran los espacios de nombres que existen:

App	Contiene las clases e interfaces básicas para el desarrollo de aplicaciones nativas
Base	Complementa a la anterior con un conjunto de tipos de datos y métodos básicos que son usados en cualquier aplicación que se desarrolle de este modo
Content	Proporciona clases e interfaces que permiten manejar la gestión de contenidos y el contenido de búsqueda
Graphics	Posibilita que las aplicaciones puedan visualizar imágenes (bitmap), polígonos y texto, gracias a las primitivas de renderizado y al soporte a OpenGL
Io	Como su nombre indica, nos permite el trabajo sobre ficheros, directorios y la manipulación de datos
Locales	Contiene lo necesario para la realización del proceso de localización e internacionalización de la aplicación para diferentes países
Locations	Posibilita que las aplicaciones puedan incorporar funcionalidades adicionales basadas en la localización del dispositivo y el acceso a servicios locales
Media	Para el control de audio, video, procesamiento de imágenes y su gestión por parte de las aplicaciones
Messaging	Para el control de todos los sistemas de envío de mensajes, ya sean estos SMS, MMS, correos electrónicos o notificaciones de tipo *Push*
Net	Para la gestión de las conexiones de red, y para el control de la transferencia de datos entre dispositivos Tizen
Security	Herramientas para el control de la seguridad como son los certificados digitales y otras clases con funcionalidad básica de criptografía
Shell	Nos proporciona herramientas para gestionar diversas características de la *shell* del terminal
Social	Facilita la creación de aplicaciones que tienen que interactuar con los datos personales del usuario sobre las redes sociales
System	Para acceder a la información y características disponibles de nuestro dispositivo
Telephony	Gestión de las llamadas, acceso a la información de la red y de la tarjeta SIM
Text	Es un *namespace* de utilidad para garantizar la interoperabilidad entre los estándares de codificación usado tanto en las aplicaciones Web, como en los SMS y MMS
UI	Son las clases básicas necesarias para la creación de la interfaz gráfica de usuario
UIx	Complementa a la anterior al proporcionar un mecanismo de acceso al uso del terminal de maneras no convencionales, por ejemplo, en este *namespaces* tenemos acceso a clases para el control de los sensores incorporados
Web	Permite integrar características web en las aplicaciones

Tabla 2.3: API para el desarrollo nativo

Cabe destacar que *Tizen OS* hace mucho hincapié en el navegador y en el estándar *HTML 5*, siendo sus versiones completamente compatibles con dicho estándar a nivel de especificación completa y con las APIs de WebRTC, permitiendo el acceso a la webcam y a la vibración del teléfono.

- *Ubuntu Touch* [27]

  Ubuntu, un viejo conocido en el mundo del software libre, prepara su llegada a móviles y tabletas. Apostando por la sencillez y ausencia de botones. Al igual que *Firefox OS* y *Tizen OS*, se basa en *HTML 5* para sacar mejor provecho de Internet. El sistema operativo de Canonical [28] es diferente al resto y ha sido diseñado para ser instalado en terminales que ya estén en el mercado. Su principal problema, al igual que ocurre en *Sailfish OS*, son las aplicaciones. El apoyo de la comunidad de desarrolladores será crucial para que el sistema operativo de Canonical despegue.

  La arquitectura es la siguiente:

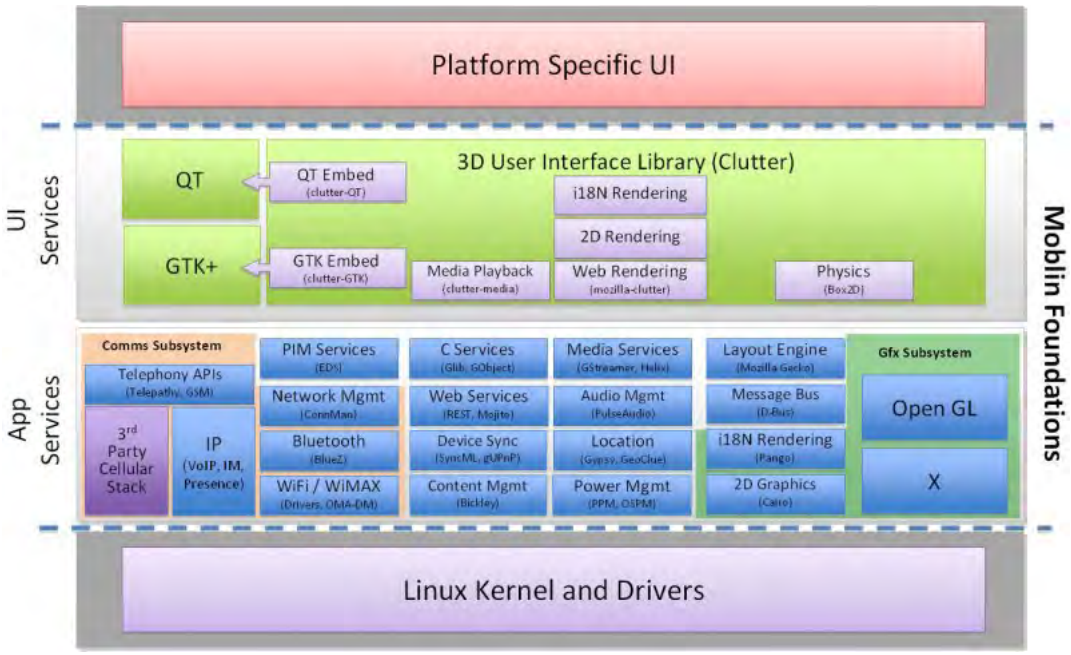


Figura 2.9: Arquitectura Sistema Operativo Ubuntu - Fuente: www.askubuntu.com

Entre las características de *Ubuntu Touch OS* destaca la capacidad para poder ejecutar aplicaciones nativas de GNU/Linux en el móvil, así como también las basadas en estándares web como *HTML 5*. Permite ejecutar aplicaciones en segundo plano y es compatible con el servicio en la "*nube*" a partir de *Ubuntu One*.

El API [29] que ofrece Canonical además de ser abierto como los anteriores, ofrece

la posibilidad de sacar lo mejor de las aplicaciones web y de las aplicaciones nativas. Ya que las aplicaciones *HTML 5* que se integran estrechamente con los dispositivos (tales como cámaras y sensores) pueden aprovechar el tiempo de ejecución de *Cordova Ubuntu* (también conocido como *PhoneGap*[9]) y el uso de su API.

Al igual que ocurría con los sistemas operativos anteriores, Canonical pone a disposición varios tipos de APIs:

- API para el desarrollo basado en Web

En la tabla siguiente se muestra la división de las diferentes funcionalidades que nos proporciona este API *UbuntuUI*:

Button	A Button
Dialog	Dialogs are modal full-screen popups that prevent other GUI interactions with the application until dismissed
Header	An Ubuntu Header wraps Tabs. Together they provide the main navigation widget at the top of an Ubuntu HTML5 app
List	A List comes with various options, including: a *header*, main text (pushed left), an icon (pushed left), and a secondary label (pushed right)
Page	An Ubuntu app consists of a Pagestack containing one or more Pages. Each page displays full-screen. See the Pagestack class
Pagestack	The Pagestack manages all Pages in a stack data structure. Initially, the Pagestack contains no Pages. The *push()* method is normally executed on load to display the app starting page
Popover	A Popover is a div containng markup that can pop up and disappear. (Unlike a Dialog, Popovers are not full screen)
Shape	An Ubuntu Shape contains and decorates (with CSS styles) some markup, often an *img*
Tabs	Tabs are the standard way to provide app navigation from your application Header. See the Header class for more information
Toolbar	A Toolbar is the JavaScript representation of an Ubuntu HTML5 app *footer*
UbuntuUI	UbuntuUI is the critical Ubuntu HTML5 framework class. You need to construct an UbuntuUI object and initialize it to have an Ubuntu HTML5 app. You then use this object to access Ubuntu HTML5 objects (and object methods) that correspond to the Ubuntu HTML5

Tabla 2.4: API UbuntuUI

Actualmente no se puede indicar nada sobre el *API Corbova Qt*, versión de código abierto de *PhoneGap*, ya que se encuentra en pleno proceso de creación.

[9] *PhoneGap [30]:* Es un *framework* para el desarrollo de aplicaciones móviles. Principalmente, permite a los programadores desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas tales como *HTML 5*, *CSS 3* y *JavaScript*. Las aplicaciones resultantes son híbridas, es decir que no son realmente aplicaciones nativas al dispositivo, pero no se tratan tampoco de aplicaciones web.

- API para el desarrollo nativo (QML)

Este API está compuesto por las librerías gráficas *Qt*, las cuáles están programadas en lenguaje *C/C++*. Además, para facilitar la programación de la interfaz se emplea lenguaje basado en *JavaScript* y en *QML* (*QT Modelling Language*).

Graphical Interface	Components.Components	Ubuntu.Components	
	Components.Components.ListItems	Ubuntu.Components.Pickers	
	Qt.labs.folderlistmodel	Ubuntu.Components.Popups	
	QtQuick	Ubuntu.Components.Themes	
	QtQuick.Particles	Ubuntu.Layouts	
	QtQuick.Window	UbuntuUserInterfaceToolkit	
	QtQuick.XmlListModel	**Other Elements**	CrossFadeImage
	Ubuntu.Components		SignalSpy
			TestCase
Platform Services	Friends	U1DB	
	QtContacts	Ubuntu.Content	
	QtLocation	Ubuntu.OnlineAccounts	
	QtOrganizer	qtpim	
Language Types	QtQml	**Other Elements**	Qt

Tabla 2.5: API QML

- *Sailfish OS* [31]

  Esta es la apuesta de antiguos ingenieros de Nokia, que toma las bases de Meego. Este sistema [32] tiene la personalización por bandera y todos los aspectos se adaptan a cualquier gusto y necesidad. Incluso es capaz de detectar el color de la carcasa para adaptar la combinación de colores del sistema operativo. Su principal problema radica en las pocas aplicaciones existentes, además del precio de los dispositivos. Sus responsables trabajan en integrar la compatibilidad con aplicaciones Android, como ya dispone Tizen. Cabe destacar que el núcleo Linux de este sistema operativo se basa en Mer.[10]

Sailfish OS proporciona diferentes APIs [33], *Qt* (*QtQuick*, *QtMobility*, *QtWebkit*,...),

[10] *Mer:* Software libre, abiertamente optimizado para *HTML 5/QML/JS*, proporcionando una distribución de base-móvil optimizado para su uso por los fabricantes de dispositivos.

así como *Cordova Qt*, soportando de este modo *HTML 5*, al igual que *Ubuntu Touch OS*. También implementa una API, la cuál se especificará en la siguiente tabla:

Simulation models	lb_base lb_single lb_binary
Geometry	geo subdomain node_type
Symbolic computation	sym
Internals	backend_cuda subdomain_runner codegen config connector controller geo_encoder master util
Visualization	vis vis_2d

Tabla 2.6: API Sailfish

- *Open WebOS* [34]

  Este sistema operativo a pasado por muchas manos y aún no ha conseguido ver la luz en lo que a dispositivos móviles se refiere. En principio, fue creado por Palm, pero debido a que no terminaba de convencer, al poco tiempo fue comprado por HP, quienes intentaron lanzarlo al mercado, pero debido a la gran competencia, optaron por instalar en sus dispositivos móviles sistemas operativos creados por otras empresas como *Windows Phone*. El no querer apostar por tabletas que tuvieran instalado *WebOS* fue una decisión de suma importancia en ese momento. Una de las últimas decisiones al finalizar el año pasado por parte de HP fue poner *WebOS* en código abierto y así desarrollar este sistema para otros dispositivos.

  Gracias a que HP dio ese paso, *WebOS* sigue a flote y, es que en realidad es un sistema operativo de gran calidad, tan solo tiene como pega la falta de aplicaciones en comparación con las creadas por la gran competencia, no obstante, al migrar el código a *open source* se detonan nuevas posibilidades de desarrollo y se deslumbran nuevas esperanzas de vida para este mismo.

  En Marzo de 2013 una noticia [35] se dio a conocer en el congreso mundial de

móviles y fue la compra de *WebOS* por parte de LG Electronics, ésta compró todo el proyecto. También HP otorgó patentes registradas y documentos oficiales que éste había heredado de Palm.

Uno de los motivos principales por lo que LG se ha comprometido a continuar con este proyecto se debe no sólo a mejorar *WebOS* para *smartphones* (siguiendo con el apoyo del *open source*) sino que también tienen en mente utilizar esta plataforma e instalarla en televisiones, las cuáles son conocidas como *Smart TV*. Con el paso del tiempo, parece ser que LG está enfocándose más hacia esos nuevos televisores que hacia los *smartphones*.

WebOS, al igual que los anteriores sistemas operativos móviles, presenta una serie de APIs [36] para programar en su entorno, éstas son las que se muestran a continuación:

Mojo App Framework APIs	Obsolet	
Enyo App Framework APIs	View the Enyo API Reference: https://developer.palm.com/content/api/reference/enyo/enyo-api-reference.html	
PDK APIs	PDK Data Types	
	Plug-in API Summary	
WebOS Application APIs	Add Contact	Messaging
	Browser	Peopler Picker
	Calendar	Phone
	Camera	Photos
	Document Viewers	View File
	Email	
	Maps	
WebOS Service APIs	Accelerometer	In-App Payment
	Activity Manager	Key Manager
	Alarms	Key Service
	Application Manager	Media Permissions
	Bluetooth	Power Management
	Connection Manager	Print Manager
	db8	System Properties
	Display Manager	System Services
	Download Manager	System Sounds
	Firewall	Touch2Share
	GPS	Zeroconf
JavaScript Library APIs	Foundations	
	Media Capture	
	Metascene.Videos	

Tabla 2.7: APIs WebOS

Después de todo el análisis de las diferentes plataformas estudiadas, la siguiente tabla indicará a modo resumen las diferencias y similitudes que pueden encontrarse entre ellas:

Fabricante	*Mozilla*	*Canonical*	*Linux Foundation*	*Open WebOS*	*Jolla*
Modelo	*Firefox OS*	*Ubuntu Touch / Phone*	*Tizen*	*Open WebOS*	OS Sailfish
Arquitecturas soportadas	*ARM*	*ARM, x86*	*ARM*	*ARM*	*ARM*
Dispositivos	*Smartphones*	*Smartphones, tabletas, PCs y televisores*	*Smartphones, tabletas, PCs, televisores y coches*	*Smartphones, tabletas y televisores*	*Smartphones*
Programado en	*C++, JavaScript, HTML5 y CSS*	*C, C++, C#, Java, Python, Vala, Qt y QML*	*HTML5, CSS, C++ y C*	*C, C++ y JavaScript*	JavaScript, QML, C++ y Qt
Motor del navegador web	*Gecko*	*Gecko*	*Webkit*	*Webkit*	*Webkit*
Familia del sistema operativo	*Linux*	*Linux*	*Linux*	*Linux*	*Linux*
VoIP/videochat	*WebRTC*	*No de serie*	*WebRTC*	*No de serie*	*No de serie*
Tienda de apps	*Firefox Marketplace*	*Ubuntu Software Center*	*No admite*	*App Catalog*	*No dispone*
Lo mejor	Basado en estándares abiertos Orientado a terminales de bajo coste para países en desarrollo	La base Open Source y las mejores prestaciones de esta distribución estarán disponibles en todo tipo de dispositivos	La combinación del soporte para aplicaciones HTML5 y nativas en diversos lenguajes es interesante para desarrolladores El apoyo de Samsung será crucial para que podamos verlo en dispositivos	Su base Open Source y su apuesta por aplicaciones web Hereda las mejores ideas de la interfaz WebOS original	Las aplicaciones Android son compatibles con este sistema operativo Permite el desarrollo de aplicaciones nativas
Lo peor	El rendimiento y capacidad de las aplicaciones HTML5 aún está por demostrarse, muchos desarrolladores siguen prefiriendo APIs nativas que parecen más potentes	Ubuntu deberá completar la transición a la arquitectura ARM Las aplicaciones nativas desarrolladas con el SDK no serán probablemente compatibles con otras distribuciones Linux	Modelo de licencias confuso La integración de Bada en él lo hace un poco más cerrado	No hay visos de que aparezcan móviles basados Es una plataforma que pretende estimular el interés de los desarrolladores que apuesten por las aplicaciones web y por frameworks como Enyo	Escasa información

Contenido extraído de: *http://www.pcactual.com*

Tabla 2.8: Tabla comparativa

Como **conclusión** sobre las APIs que se detallaron anteriormente, se demuestra que algunas de ellas son equivalentes en cuanto a funcionalidad. Pero internamente cada sistema operativo móvil emplea su metodología, en el caso de *Firefox OS* todas las APIs siguen un modelo simple: el usuario pide el acceso y se define un controlador de éxito y otro de fracaso; en el caso de *Tizen OS*, las APIs definen dos tipos de funciones, de la misma manera que *Firefox OS*, una de éxito y otra de fracaso, pero cabe añadir que cuando se inicia la aplicación, se crea automáticamente una instancia del tipo *SystemInfoObject* en

el objeto Tizen. Siendo éste, *tizen.systeminfo*, una instancia de la interfaz *SystemInfo*, que proporciona acceso a la información del sistema [37]. Si observamos las APIs de *WebOS*, vemos que ocurre algo parecido, y es que en alguna de las APIs existen esas dos funciones aunque luego tengan su metodología particular como sistema operativo propio.

En el caso de *Ubuntu Touch OS*, emplea APIs similares a *Sailfish OS* (*Qt*), por lo que en ese aspecto, tienen bastante en común ambos y es algo que les diferencia de los demás. Aun así, todos ellos tienen APIs basadas en Web, unos a partir de *QT Cordova* y otros mediante *HTML 5* directamente.

2.2. Requisitos

Los requisitos establecidos para la realización de este proyecto tienen que ver con las diversas plataformas estudiadas. Se requiere que las aplicaciones que se creen funcionen, tanto en el simulador como en el dispositivo *Firefox OS*, cuya interfaz sea sencilla, cómoda y manejable para que su experiencia con este nuevo entorno sea de fácil inmersión; y, a su vez, estas aplicaciones serán probadas en algunos sistemas operativos móviles (aquellos que difieran de *Firefox OS*) con el fin de obtener unos resultados concluyentes que pondrán fin a dicho estudio.

Otro de los aspectos a tener en cuenta es que las pruebas serán llevadas a cabo posteriormente en otros sistemas operativos móviles como se comentó anteriormente, es decir, serán efectuadas en sus respectivos entornos, por lo que si se prescinde de los dispositivos se emplearán los emuladores correspondientes.

2.3. Restricciones y Marco Regulador

Para la implementación de la solución requerida por el cliente, éste ha solicitado que no se empleen SDKs cuyo uso implique el pago de licencias, no importándole si se trata de código libre o no, mientras éste sea gratuito. Por lo tanto, se comprueba que los SDK´s

que se emplean cumplen esta restricción, ya que se trata de soluciones de código abierto, es decir, no es preciso pagar una licencia para poder utilizarlo.

La intención del cliente es poner a disposición de la Comunidad el estudio de las diferentes plataformas una vez creadas las aplicaciones, dando una mayor información a los desarrolladores para que éstos puedan mejorar algunos aspectos en sus aplicaciones y que éstas sean lo más compatibles posibles.

Salvo *Tizen OS*, que su SDK completo se ha publicado bajo la licencia de Samsung, ninguna de las demás plataformas estudiadas impone una serie de restricciones a las aplicaciones para que puedan ser publicadas en su sistema de distribución, por lo que sus comunidades no tienen en cuenta diversas restricciones que tienen que ver con aspectos de diseño o éticos. En el caso de Tizen, que es el que más límite impone en este sentido, se ha comprobado que para el estudio llevado a cabo, la simulación de las aplicaciones en ese entorno no conlleva pago de licencia alguno.

Finalmente y a modo de resumen, decir que, los diferentes Términos y Condiciones ofrecidos por las plataformas estudiadas conforman los requisitos y el marco regulador del presente proyecto.

Capítulo 3

Diseño de la solución técnica

3.1. Procedimiento

Dado que el objetivo de este estudio consiste en realizar diversas comprobaciones en sistemas web que repercuten al área multimedia, a continuación se indican las aplicaciones que se han desarrollado para dicho fin, indicándose en el capítulo posterior, las conclusiones obtenidas de dicho estudio.

Para ello, la mejor forma de empezar a comprender las características de cualquier nuevo entorno de desarrollo es mediante la creación de un ejemplo sencillo. En este caso, y como viene siendo habitual, se creará una aplicación, de cada uno de los sistemas operativos móviles que se van a estudiar, que mostrará por pantalla el mensaje "Hola Mundo".

Después el estudio se encaminará hacia otros sistemas operativos que tienen algo más de apoyo como es el caso de *Firefox OS*. Por lo que a la hora de implementar las aplicaciones esta plataforma toma el relevo ya que además se dispone de dispositivo. Posteriormente, se expondrán las aplicaciones que han sido creadas en el sistema operativo Firefox así como las APIs que se han empleado en éstas, dichas aplicaciones serán probadas (en el capítulo posterior) en los sistemas operativos móviles que difieren de éste, como por ejemplo, el motor del navegador, y serán evaluadas también en dicho capítulo para así poder concluir el estudio.

Cabe destacar, que estas aplicaciones no se prueban en todos los sistemas operativos debido a que si se estudian las plataformas que más se asemejan y difieren de *Firefox OS* estarán todos los sucesos estudiados, ya que son casos extremos. Además, como se detalló en el capítulo anterior, prácticamente todas las APIs son equivalentes en cuanto a funcionalidad por lo que no habrá problema alguno en este caso.

En cuanto a las aplicaciones que se crean en este entorno (Firefox), una es un poco más sencilla, ya que se desea que no requiera acceso alguno al *hardware* del dispositivo, denominada *ColorLight*; y otra que, al contrario que la anterior, debe tener acceso al mismo, *Images ScanApp*. De esta manera, el estudio se podrá enfocar mejor, ya que se puede diferenciar los límites y ventajas que impone cada una de ellas en los demás entornos.

Por lo que, como se a comentado anteriormente, el estudio será más completo, ya que se evaluará en el capítulo posterior, cómo las funciones de una de las aplicaciones se verán limitadas en otros sistemas operativos móviles por el hecho de tener que recurrir a funciones nativas para acceder al *hardware*, y la otra, en cambio, no.

3.1.1. HolaMundo en WebOS

Para comenzar, una vez leída la documentación que hace referencia a este entorno [38], se observa que existen dos "modos" para crear aplicaciones, una es a partir del software *Eclipse* y la otra, a partir del terminal de Linux (mediante línea de comandos).

Indicar que si se crea la aplicación mediante línea de comandos, será necesario iniciar la aplicación, empaquetarla e instalarla en el emulador de manera manual siempre que se desee ejecutar, teniendo en cuenta que para posteriores simulaciones tendrá que estar ésta cerrada y eliminada, para que se puedan efectuar los cambios. Mientras que en *Eclipse* todo este proceso se realiza de manera automática. De todos modos, ambos métodos se pueden ver de manera más detallada en el Anexo B.1.

Figura 3.1: Aplicación HolaMundo ejecutándose en Simulador WebOS

Destacar que si la aplicación se realiza desde *Eclipse*, se crean algunos archivos más por defecto que los que se crean básicos desde línea de comandos. A continuación se muestran los archivos que se crean en cada método[1]:

Método 1: Línea de comandos -APP_DIR-

- *Carpeta de imágenes* - Contiene las imágenes que esta aplicación emplea.
- *Carpeta de hojas de estilo (CSS) y JavaScript* - Contiene archivos referentes a hojas de estilo en cascada que utiliza la aplicación, así como archivos de tipo *JavaScript*.
- *appinfo.json* - Archivo que contiene información de configuración importante sobre la aplicación.
- *depends.js* - Archivo que direcciona los ficheros de tipo *CSS* y *JavaScript*.
- *framework_config.json* - Archivo que contiene la configuración del marco.
- *index.html* - Escenario principal en el que aparecerán las escenas de la aplicación.

Método 2: Eclipse -workspace-

- *Carpeta de la aplicación -app-* - Contiene los asistentes, modelos y vistas que componen la aplicación. Más adelante en su respectivo Anexo se agregarán archivos a este directorio.

[1] *NOTA:* Véase Anexo B.1 para más información.

- *Carpeta de imágenes* - Contiene las imágenes que esta aplicación emplea.
- *Carpeta de hojas de estilo (CSS)* - Contiene archivos referentes a hojas de estilo en cascada que utiliza la aplicación.
- *appinfo.json* - Archivo que proporciona información que el marco SDK utiliza para empaquetar y ejecutar la aplicación.
- *framework_ config.json* - Archivo que contiene la configuración del marco.
- *icon.png* - Imagen de inicio que se muestra en el emulador o dispositivo.
- *index.html* - Escenario principal en el que aparecerán las escenas de la aplicación.
- *sources.json* - Lista de los archivos de origen para cada escena.

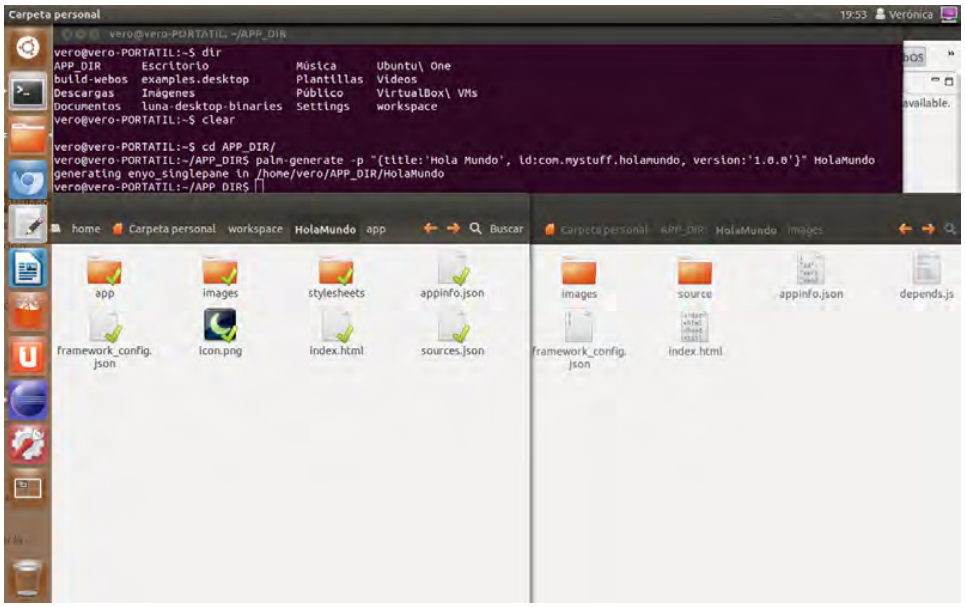


Figura 3.2: De izquierda a derecha se muestra el contenido de los directorios workspace y APP_DIR

Destacar que se procedió a estudiar este sistema operativo, pero dado que disponía de muy pocas APIs para acceder al *hardware* (no disponía de acceso al Bluetooth, a las conexiones de red,...), no se ha llegado a profundizar en él como se esperaba en un principio. Además, esta plataforma no a tenido mucho empuje en lo referente a entornos

móviles, ya que LG comenzó a interesarse por otras tecnologías claves del mercado como son los *Smart TV*.

3.1.2. HolaMundo en Firefox OS

En el caso de *Firefox OS*, dado que la creación de una aplicación es más sencilla que en *WebOS*, a continuación se muestra cómo se a creado ésta, así como los ficheros de los que está compuesta esta aplicación básica.

Para crearla requiere que el navegador de Firefox no supere la versión 25, ya que el simulador *Firefox OS Simulator*[2], no es del todo compatible con Firefox 26+. Este simulador está empaquetado y distribuido como un complemento de Firefox. En él se lanzarán las aplicaciones y se depurarán. También cabe indicar que, dado que para versiones posteriores a la 26, como es el caso en estos momentos, Mozilla aconseja emular las aplicaciones en otro simulador creado por ellos llamado *App Manager*[3]. Éste requiere que se instale su simulador, así como el *ADB Helper* en caso de que se deseen hacer las pruebas en el dispositivo. En este momento, se mostrará en ejecución la aplicación desde el simulador *App Manager*, ya que en posteriores capítulos se visualizarán otras aplicaciones más interesantes para el estudio desde ambos.

NOTA: *Destacar que este simulador es compatible con sistemas Windows, Mac OS y Linux.*

También hará falta un editor de texto plano, ya sea *Dreamweaver, Notepad ++, Eclipse,...*En este caso, se hará uso de *Adobe Dreamweaver CS6*.

Antes de comenzar a indicar los tipos de ficheros que se deben crear para que funcione una aplicación en *Firefox OS*, habría que destacar que existen dos tipos de aplicaciones en este entorno, **empaquetadas y alojadas**. Las primeras consisten en un archivo *zip* que contiene todos los recursos de la aplicación, archivos *HTML*, *CSS*, *JavaScript*, imágenes,

[2] *Firefox OS Simulator:* Compatible con versiones del navegador de Firefox 24/25 y Firefox OS 1.1.
[3] *App Manager:* Compatible con versiones del navegador de Firefox 26+ y Firefox OS 1.2.

tipografías,...mientras que las segundas consisten en una aplicación que se aloja en un servidor web, como cualquier otra página.

Para que funcionen estas aplicaciones, debe haber un archivo *manifest* válido. El archivo *manifest.webapp* contiene toda la información que un navegador web necesita para interactuar con una aplicación. Un *manifest* es una de las cosas claves que distinguen una *Web App* de un sitio web. Se trata de un archivo *JSON* con un nombre y una descripción para la aplicación, y también puede contener el origen de la aplicación, iconos, y los permisos requeridos por la aplicación, entre otras cosas.

Por lo que, en este caso, además de añadir el *manifest*, se toma una imagen como icono, ésta debe de tener como dimensiones 128x128. Y, por último, en un fichero *HTML*, se crea esta sencilla aplicación. Cabe indicar que para aplicaciones que requieran más dificultad e interacción con el usuario se requerirá el uso de ficheros *JavaScript* y *CSS* como se podrá comprobar en aplicaciones posteriores.

A continuación, se muestra cómo se visualiza la primera aplicación creada en este entorno con el simulador *App Manager*, ya que en el Anexo C.1 se ejecuta en ambos simuladores. Además, se puede consultar éste para obtener más detalles de cómo crear la aplicación.

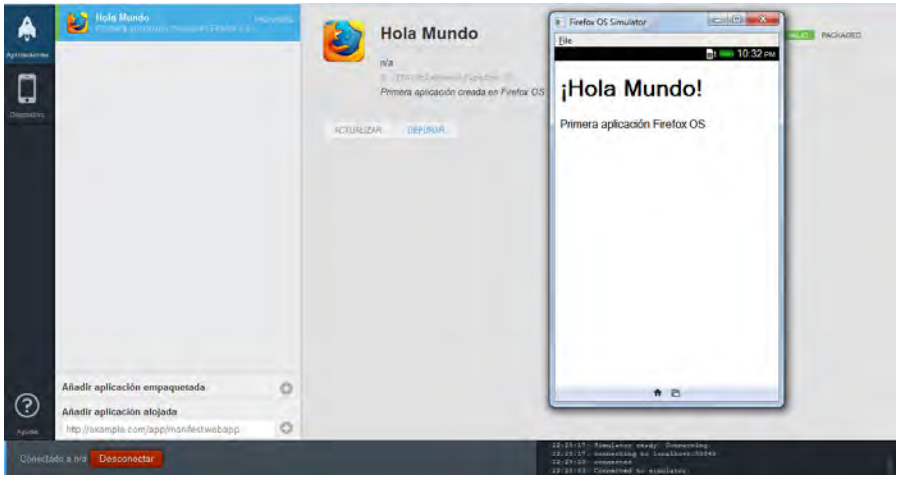


Figura 3.3: Emulando aplicación en el simulador App Manager

3.1.3. HolaMundo en Tizen OS

Al instalar el SDK de Tizen, se da la opción de seleccionar que ésta sea mediante instalación de red o mediante instalación de imágenes SDK. En este caso, se elige el segundo método de instalación, ya que el archivo de imagen SDK se encuentra almacenado en un paquete que contiene todos los archivos necesarios para un entorno sin conexión. De todos modos, en su web [39] detallan cómo instalarlo de ambas maneras.

Cabe indicar que dependiendo del sistema operativo donde de ejecute el simulador, hay que instalar el SDK de un modo u otro. Destacar que en el Anexo D.1 se contemplan los diversos modos de instalación y, además, se puede ver de manera más detallada todo lo referente a la creación de esta aplicación en este nuevo entorno.

A continuación se muestra la aplicación ejecutada en el simulador:

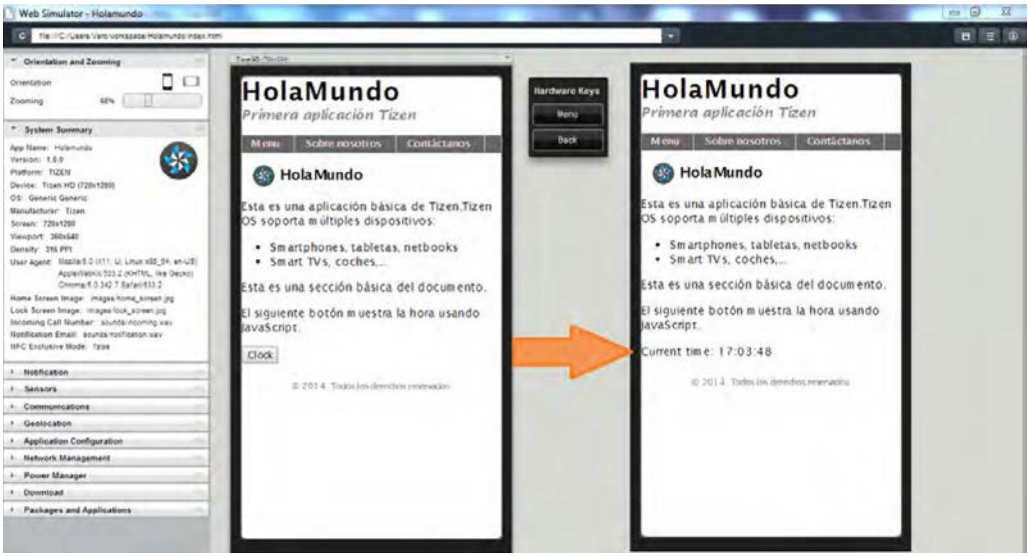


Figura 3.4: Ejecutando la aplicación Holamundo en el Simulador de Tizen OS

En cuanto al empaquetado de la aplicación, dependiendo de la técnica que se emplee a la hora de desarrollar en *Tizen OS*, ésta se empaquetará de un modo u otro. En esta plataforma existen tres modos de desarrollo, los cuáles se exponen a continuación:

1. Desarrollo basado en estándares web (*HTML 5, CSS 3 y JavaScript*).

2. Desarrollo nativo (*C* y *C++*).

3. Desarrollo híbrido, en donde se combinan los dos anteriores para crear una aplicación.

En este caso se empleará el primer modo de desarrollo, por lo que el tipo de paquete que se obtendrá será el ***Tizen Web Package***[4]. Este paquete funciona como un fichero zip pero con extensión ".wgt" y se basa en el estándar de la W3C para el empaquetado y configuración de las aplicaciones. Indicar que si el modo de programación fuese nativo (que no es el caso), el tipo de paquete que se crearía sería del tipo ***Tizen Package***, funcionando al igual que el anterior como un fichero zip, pero en este caso con extensión ".tpk".

Cabe destacar que el *Package Manager* o Administrador de Paquetes es uno de los componentes básicos del *framework* de aplicaciones y es el responsable de realizar la instalación, desinstalación y cualquier modificación en aquellos paquetes que estén instalados en el dispositivo[5].

Además, el *Tizen Web Package* sigue el formato de W3C (como se comentó anteriormente) en cuanto a ficheros y estructura de directorios. Por lo que en el caso de la aplicación "HolaMundo", la estructura de directorios que la aplicación tiene es similar a la siguiente:

- *config.xml* - Fichero de configuración.
- *icon.png* - Icono de la aplicación.
- *index.html* - Fichero de arranque de la aplicación.
- *Holamundo.wgt* - Paquete de la aplicación "HolaMundo".
- *.project* - Archivo en el que se almacena la configuración del proyecto.

[4] *Tizen Web Package:* Este tipo de paquete se genera en las aplicaciones híbridas y en las basadas en los estándares web.

[5] *URL Web:* Si se desea obtener más información, consultar: https://developer.tizen.org/es/downloads/sample-web-applications/load-web-app-tizen-sdk

- *images_tizen_32.png* - Imagen que se emplea en la aplicación.
- *css_style.css* - Fichero que incluye el estilo de la aplicación.
- *js_main.js* - Fichero de texto plano que contiene scripts de *Javascript*, es decir, contiene la parte lógica de la aplicación.

3.1.4. HolaMundo en Sailfish OS

Sailfish OS ofrece también su SDK, el cuál puede ser ejecutado en los tres sistemas operativos por excelencia (Linux, Windows y Mac OS). En el Anexo E.1 se indicará con más detalle cómo se debe instalar dicho SDK según el sistema operativo que se emplee, y se indicarán los pasos que se han de seguir para crear la aplicación "HolaMundo" en este entorno. A continuación se mostrará ya la aplicación ejecutada en el simulador:

Figura 3.5: Simulando aplicación Holamundo en Sailfish OS

NOTA: *Si durante la simulación de la aplicación se muestra una pantalla en negro, se debe a que el emulador se piensa que es un dispositivo real, por lo que aplica el ahorro de energía. Para quitar esta pantalla basta con mover el ratón haciendo un gesto hacia abajo o emular el botón de inicio del dispositivo pulsando "Host + 'H"'.*

Cuando se crea esta aplicación, los ficheros que se generan de manera automática son los siguientes:

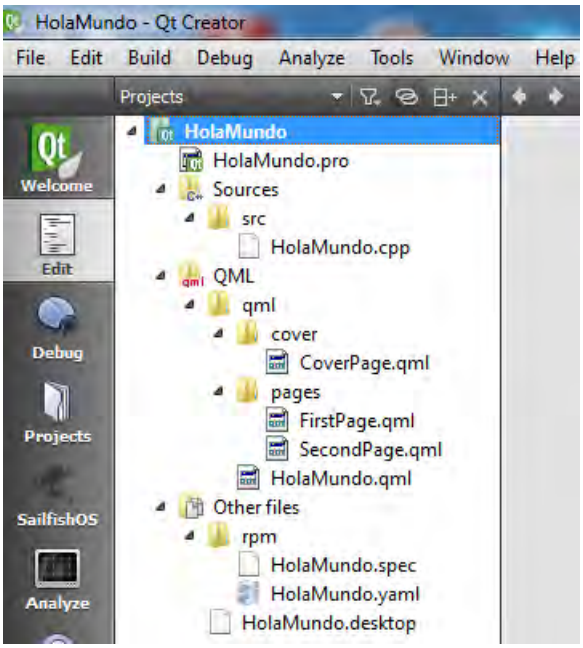


Figura 3.6: Ficheros que conforman la aplicación Holamundo en Sailfish OS

A continuación se indica qué es cada fichero así como la extensión que han de tener:

- *HolaMundo.pro* - Es el proyecto de la aplicación, el cuál une el código fuente y el binario.
- *Sources/src/HolaMundo.cpp* - Implementa el punto de entrada a la aplicación simplemente pasando el recuento de argumentos a la función *main()*. Dicha función espera el nombre del archivo *QML* que tiene asociado el nombre del proyecto, en este caso "HolaMundo.qml", por lo que será el primer archivo que cargue la aplicación.
- *QML/qml/cover/CoverPage.qml* -Es la cubierta de la aplicación.
- *QML/qml/pages/FirstPage.qml* -Es la primera página de la aplicación.
- *QML/qml/pages/SecondPage.qml* -Es la segunda página de la aplicación.
- *QML/qml/HolaMundo.qml* -Es la base de la aplicación, la que indica cuál es la primera página de ésta así como su cubierta.

- *Other files/rpm/HolaMundo.spec* -Es el fichero que indica cómo construir, instalar y desinstalar el paquete de la aplicación.

- *Other files/rpm/HolaMundo.yaml* -Es un fichero en el que se indica los archivos de configuración de la aplicación.

- *Other files/HolaMundo.desktop* -Es un archivo de configuración de escritorio estándar de Linux, el cuál describe cómo se va a poner en marcha la aplicación, el nombre de ésta, la forma en que aparecerá en los menús, el icono de la aplicación (que será de dimensiones 86x86), etc.

3.1.5. HolaMundo en Ubuntu Touch OS

El SDK de este nuevo entorno, al igual que ocurre con los SDK de Sailfish y Tizen, proveen de un Entorno de Desarrollo Integrado (IDE) por lo que de este modo, la creación de aplicaciones será más amena, rápida y eficaz. La diferencia que tiene con estos últimos es que el SDK de esta plataforma requiere ser ejecutado, en Linux únicamente (al igual que ocurre en WebOS), en este caso se emplea más concretamente Ubuntu como sistema operativo.

En cuanto al empaquetamiento se refiere, por el momento las aplicaciones seguirán la extensión del formato de paquetes de software de la distribución de Linux Debian y derivadas, como es el caso de Ubuntu, por lo que se empaquetará en .deb. Aunque puede que este tipo de formato de empaquetamiento en un futuro se vea modificado para aquellos dispositivos móviles que se basen en Ubuntu, pasando a ser "*Click Packages*" los cuáles emplearán un sistema de gestión distinta a la actual.

Sobre la aplicación "HolaMundo", la ejecución es la siguiente[6]:

[6] *NOTA:* Para obtener información detallada de cómo crear esta aplicación, véase el Anexo F.1

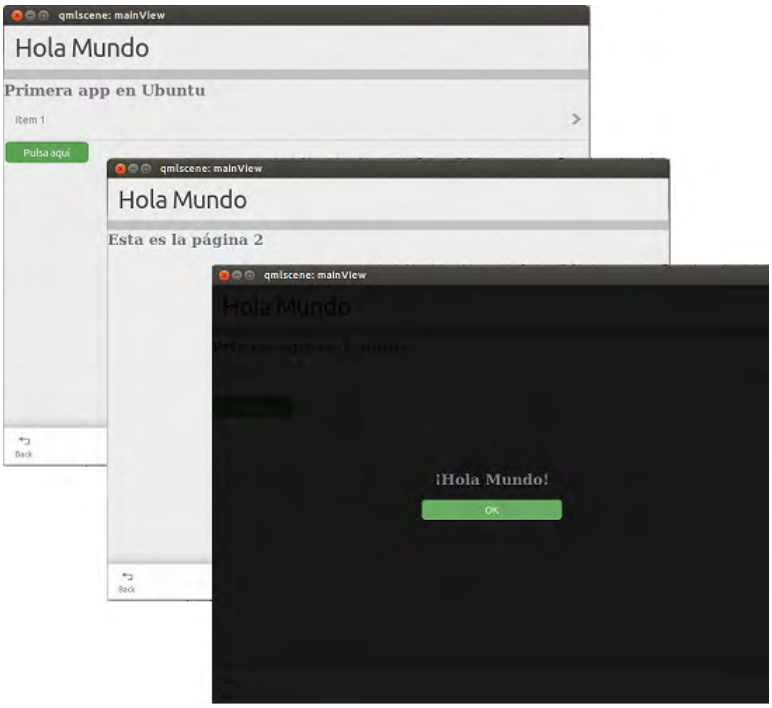


Figura 3.7: Simulando aplicación HolaMundo en Ubuntu Touch OS

Una vez se ha creado ésta, los ficheros que se generan de manera automática son:

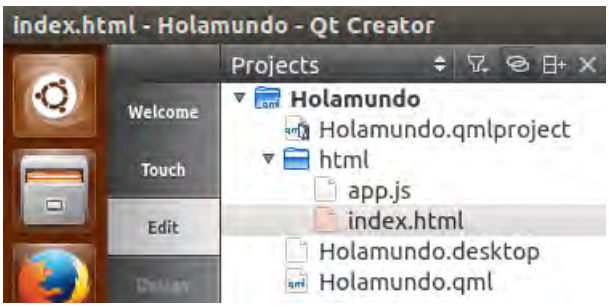


Figura 3.8: Ficheros que conforman la aplicación Holamundo en Ubuntu Touch OS

- *Holamundo.qmlproject* - Es el proyecto de la aplicación.
- *html/index.html* - Es el contenido en html de la aplicación.
- *html/app.js* - Es el contenido en *JavaScript* de la aplicación.
- *Holamundo.desktop* -Es un archivo de configuración de escritorio estándar de Linux, el cuál describe cómo se va a poner en marcha la aplicación, el nombre de ésta,

la forma en que aparecerá en los menús, el icono de la aplicación (que será de dimensiones 86x86), etc.

- *Holamundo.qml* -Es la base de la aplicación, la que indica desde dónde debe de comenzar la aplicación, en este caso, y en la mayoría desde "index.html".

Cabe destacar que el directorio *html* contiene archivos que deben ser desplegados como parte de su solicitud. Por lo que los archivos del tipo audio, imagen y archivos de *JavaScript* que se deseen emplear en aplicaciones futuras, deberán ir en este directorio.

3.1.6. ColorLight en Firefox OS

ColorLight simula a una linterna como la que se puede encontrar en cualquier *smartphone* de hoy día. Aunque en este caso, se empleará únicamente la pantalla, ya que el API de *HTML 5* no contempla el acceso al *flash* del dispositivo. De hecho, el primer *smartphone* que funciona con sistema operativo Firefox, bautizado como ZTE Open, no dispone ni siquiera de él. Más adelante se probará a instalar la aplicación en él para comprobar su funcionamiento.

En cuanto a la aplicación, *ColorLight* consta de tres opciones de funcionamiento las cuáles son similares a las aplicaciones que se presentan en dispositivos Android e iOS; la primera, que ofrece la posibilidad de encender y apagar la linterna; la segunda, que muestra una paleta de colores en los que el usuario podrá elegir el color deseado a la hora de que se encienda ésta, así como activar el modo aleatorio para que cada vez que se encienda el color varíe de manera aleatoria; y por último, está a disposición el modo de emergencia, en el que la pantalla comienza a parpadear de manera constante.

Figura 3.9: Icono ColorLight

Señalar que esta aplicación, *ColorLight*, está desarrollada mediante los estándares *HTML*, *CSS* y *JavaScript*, además, en ella se emplea la librería de *JavaScript* por excelencia,

jQuery. Se decide emplear este *framework* porque de este modo ayuda a economizar el tiempo y, sobre todo, porque hace que la aplicación sea más eficaz. Cabe destacar que en esta aplicación se emplea el API *Screen.lockOrientation* para fijar la orientación de la aplicación a modo vertical.

En cuanto a la funcionalidad, cabe indicar que la aplicación es multiplataforma, pudiendo ser desplegada sobre cualquier navegador (independientemente del sistema operativo), siempre y cuando ésta emplee los estándares mencionados anteriormente *(HTML, CSS y JavaScript)*, indistintamente de si el navegador es de tipo *Webkit* o *Gecko*. En el capítulo posterior se realizarán las pruebas necesarias para corroborar que esto es así.

En un primer momento, esta aplicación se ha ido emulando para comprobar su funcionamiento con ambos simuladores. A continuación se muestra la aplicación en plena ejecución desde el simulador que se encuentra como complemento de Firefox, pues en el capítulo 4.1 se mostrará cómo se ejecuta ésta en ambos:

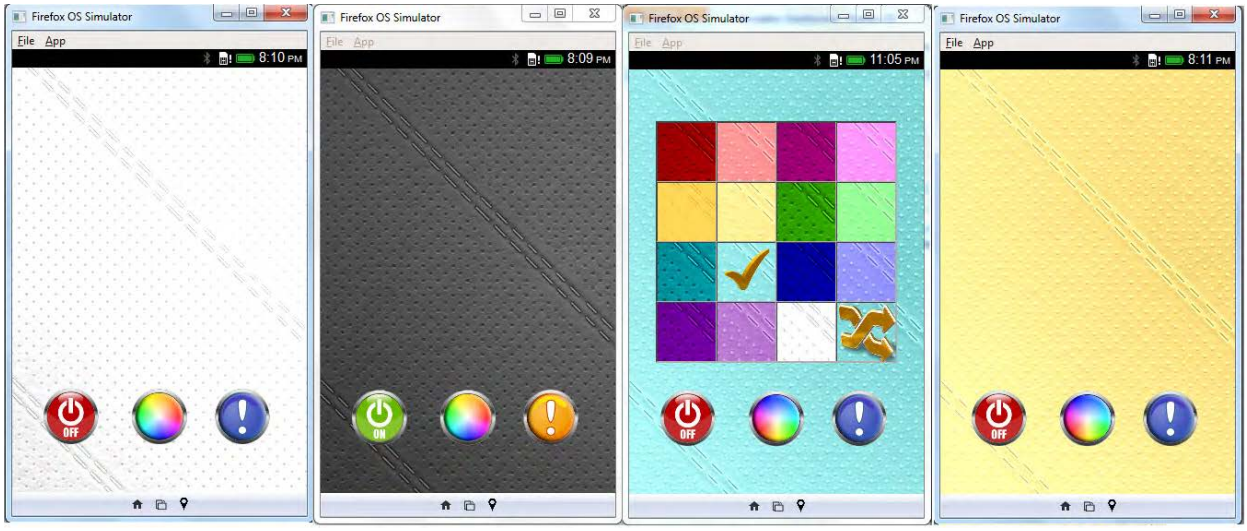


Figura 3.10: Emulando Aplicación ColorLight con el simulador que está como complemento de Firefox

3.1.7. Images ScanApp en Firefox OS

Esta aplicación además de emplear los lenguajes estándar -*HTML*, *CSS* y *JavaScript*- emplea parte de la *WebAPI*[7], concretamente *Web Activities* y *Device Storage API*. Las *Web Activities* son actividades web que permiten que la aplicación delegue de una actividad a otra aplicación en el dispositivo, en este caso, será la actividad de la cámara la que se mostrará cuando se recurra a este API. En el caso de *Device Storage API*, se emplea ésta porque trata sobre el almacenamiento en el dispositivo. Por lo que en este caso, se utiliza para acceder al sistema de archivos -galería y *wallpapers*- dentro de la aplicación web *Images ScanApp*.

Figura 3.11: Icono Images ScanApp

Destacar que al igual que la anterior aplicación (*ColorLight*), en ésta también se decide emplear el *framework jQuery*. Pero en este caso, se utiliza porque se requiere el uso de *jQuery Mobile* para mejorar la interfaz de la aplicación, ya que se trata de un sistema de interfaz de usuario diseñado para los dispositivos móviles. Pero dado que se construye sobre la base de *jQuery*, es necesario tener instalado éste para que *jQuery Mobile* funcione. Cabe indicar que en esta aplicación se emplea también el API *Screen.lockOrientation* para fijar la orientación de la aplicación a modo vertical.

Images ScanApp es una primera aproximación de una aplicación que hace de "*scanner*", con ella se pretende que el usuario pueda realizar las instantáneas que desee y pueda también modificar el estilo de éstas, recortándolas y aplicándolas los efectos que crea conveniente mediante el uso de filtros que siguen con el carácter web de *Firefox OS* ya que se crean a través de *CSS* y *SVG*. También se puede comprobar el espacio que hay disponible en el dispositivo así como el espacio que ocupan las imágenes.

[7] *WebAPI:* Es esencialmente una *interface* de *JavaScript* que permite el acceso a funciones nativas (administrar contactos, acceder a la cámara, enviar mensajes SMS, notificaciones,...)

Para la simulación de ésta se ha procedido del mismo modo que la aplicación anterior, empleando como simuladores el complemento del navegador de *Firefox OS* y el *App Manager*. A continuación se muestra la aplicación en plena ejecución desde el primero de los simuladores, pues en el capítulo 4.1 se mostrará cómo se ejecuta ésta en ambos.

Figura 3.12: A la izquierda se muestra la pantalla principal de la aplicación, en el centro y a la derecha se visualiza la información que proporciona el simulador y/o dispositivo cuando se pulsan los respectivos botones (Espacio Disponible - Espacio Ocupado).

Capítulo 4

Evaluación y Resultados

4.1. Evaluación

Durante el proceso de desarrollo, el procedimiento seguido para evaluar el correcto funcionamiento de las aplicaciones referentes a *Firefox OS* ha sido el ir ejecutando éstas en el simulador que proporciona el navegador de este entorno, así como en el *App Manager* y, posteriormente en el dispositivo para comprobar que el código escrito funcionaba correctamente. Por tanto, la evaluación durante el desarrollo inicial ha consistido en ir programando y comprobando lo programado sobre los simuladores y sobre el dispositivo físico.

Destacar que se ha efectuado en ambos simuladores ya que dependiendo de la versión del navegador, uno es compatible y otro no. Por lo que para tener una mayor seguridad de que las aplicaciones funcionan bien, se decide comprobarlo en los dos.

Llegando a las fases finales de desarrollo, las aplicaciones fueron probadas en el sistema operativo *Ubuntu Touch OS*, ya que es la plataforma que más se asemeja a *Firefox OS* en cuanto a características, y en *Tizen OS*, que, al contrario que Ubuntu, es el entorno que más diferencias tiene con respecto a *Firefox OS* (véase la tabla comparativa 2.8). De este modo, se podrá obtener mejores conclusiones en el estudio que se está llevando a cabo. Además, se ha añadido a éste (dado que se dispone de dispositivo) el sistema operativo

Android, para comprobar hasta dónde pueden llegar las aplicaciones creadas y, como una de las ventajas de este tipo de aplicaciones es que suelen ser multiplataforma, ésta será una característica que se lleve a estudio.

Por lo que, en definitiva, las aplicaciones serán probadas en aquellos sistemas operativos móviles que tenían diferentes configuraciones con el fin de llegar a unas conclusiones coherentes y dar por finalizado el estudio. Porque como se comentó en el anterior capítulo, no hace falta que estas aplicaciones sean probadas en todos los sistemas operativos ya que si se evalúan las plataformas que más se asemejan y difieren de *Firefox OS* estarán todos los casos englobados. Además, como se detalló en el capítulo 2.1, prácticamente todas las APIs son equivalentes en cuanto a funcionalidad por lo que no habrá problema alguno en este caso.

En el apartado que se muestra a continuación, se indicarán los resultados que finalmente se han obtenido en la consecución del presente trabajo fin de grado.

4.2. Resultados

Una vez completado el proceso de desarrollo y evaluación de las aplicaciones, se procede a indicar las funcionalidades y restricciones de las APIs que se han empleado. Después, se indicará de una manera más detallada las pruebas realizas así como los resultados obtenidos durante todo el estudio.

4.2.1. APIs empleadas

Cabe destacar, que en el caso de las APIs de *Firefox OS* que son las que se han empleado, se puede observar que algunas de las que proporciona la *WebAPI* no están estandarizadas aún. Esto se debe a que es un proyecto joven, y algunas de éstas aún están en pleno proceso de estandarización. De hecho, esta información está corroborada por Soledad Penadés (Ingeniera Senior, empleada Mozilla) quien me indica expresamente que

"*Las APIs se están enviando a W3C para que se estandaricen y lo implementen todos los navegadores*".

En concreto, en la aplicación *Images ScanApp* que se necesitaba acceder a la cámara de algún modo, se observó que las únicas APIs de las que disponía *Firefox OS* para realizar esta tarea no estaban estandarizadas aún. De hecho en una de ellas, *API Camera*, se indica expresamente: "*Note: Except if you are implementing a replacement for the default Camera application, you shouldn't use this API. Instead, if you want to use the camera in your device, you should use the Web Activities API.*". Por lo que, respetando la nota, se decidió emplear *Web Actitivies API*.

Pero no todo son inconvenientes, hay que mencionar que el empleo de *Web Actitivies API* es realmente cómodo y muy fácil de asimilar. Por lo que se puede lanzar cualquier tipo de actividad[1] (Galería, Ajustes, Contactos, Llamadas, Email, SMS, Cámara, Vídeos, Música,...) de una manera rápida y sencilla.

En cambio, con *Device Storage API*[2] no existen problemas de estandarización, pero sí impone ciertas restricciones a la hora de que ésta se pueda emplear, estando disponible únicamente si el tipo de aplicación es de tipo privilegiada o certificada. Pero este tipo de limitación es lógica debido a que este API accede al interior del teléfono. También hay que señalar que tiene multitud de funcionalidades, se puede acceder al teléfono incluso para comprobar el espacio que ocupan los ficheros multimedia así como comprobar el espacio que le queda libre al dispositivo.

Otra de las APIs que se emplean, y ésta además se utiliza en ambas aplicaciones porque tanto en una como en otra se desea que la orientación de la aplicación se mantenga

[1] *NOTA:* Para más información acerca de este API ir a https://developer.mozilla.org/en-US/docs/WebAPI/Web_Activities

[2] *Device Storage API:* Para más información acerca de este API ir a https://developer.mozilla.org/en-US/docs/WebAPI/Device_Storage

en vertical ("*portrait*"), es el API *Screen.lockOrientation*[3]. Cabe indicar que este API se encuentra actualmente en fase de proyecto. Por lo que sólo se implementa como un método prefijado (*mozLockOrientation*) en B2G y Firefox para Android.

4.2.2. ColorLight

Como se comentó en el capítulo anterior, esta aplicación a pasado por varias pruebas en diferentes emuladores y dispositivos con el fin de llegar a unas conclusiones sólidas. A continuación se indican éstas:

Pruebas realizadas en emulación

- *Simulador Firefox OS -Complemento-*

  En primer lugar, para ir comprobando el funcionamiento de la aplicación, mientras se desarrollaba ésta se iba comprobando en el simulador. Para ello se añade el archivo de tipo *manifest* como se muestra en la figura para ejecutar la aplicación:

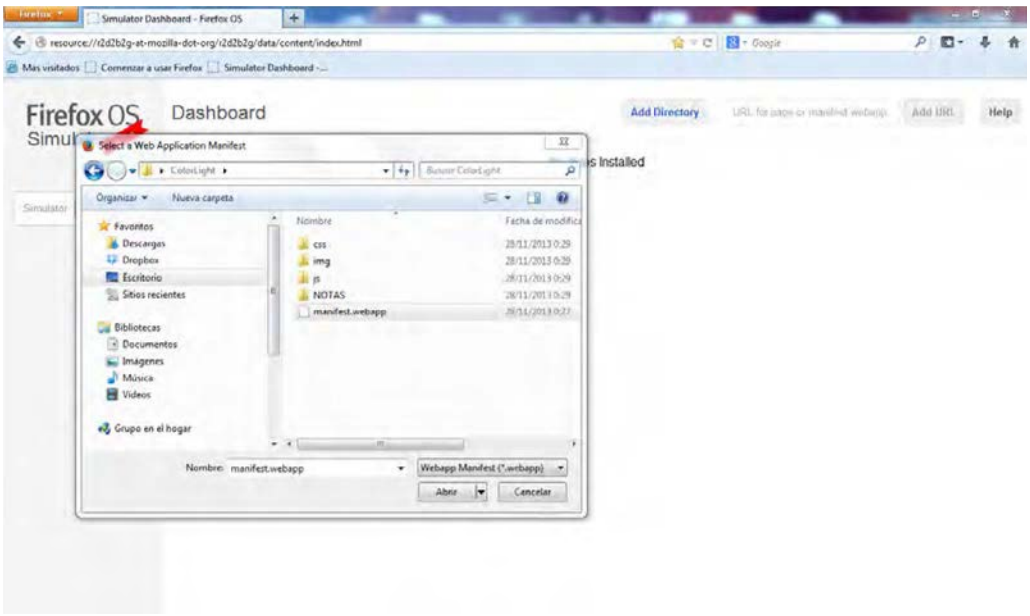


Figura 4.1: Añadiendo la aplicación al simulador

[3] *API Screen.lockOrientation:* Para más información acerca de este API ir a https://developer.mozilla.org/en-US/docs/WebAPI/Detecting_device_orientation

Éste es simplemente un archivo *JSON* que contiene los metadatos de la aplicación, siempre tiene que encontrarse en la raíz de la aplicación (en el mismo directorio donde se encuentran los archivos de tipo *.html*) y debe llamarse *manifest.webapp*. Una vez se ha añadido la aplicación, se ejecuta el emulador para comprobar si el resultado es el esperado o no:

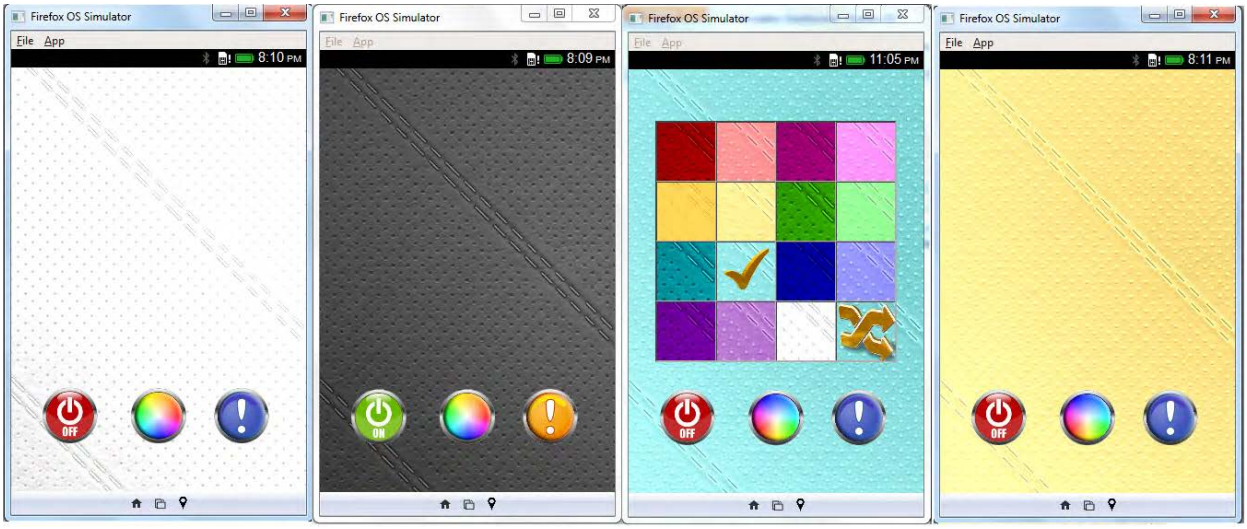


Figura 4.2: Emulando Aplicación ColorLight con el simulador que está como complemento de Firefox OS

Cabe indicar que este simulador, según Mozilla, sólo es compatible con versiones del navegador de Firefox 24 y 25 y, para que se puedan portar las aplicaciones al dispositivo, se requiere que éste se encuentre en la versión Firefox OS 1.1. En este caso, teniendo recientemente la versión 26 de navegador, hay que recalcar que la aplicación funciona según lo esperado y se puede portar perfectamente al dispositivo, el cuál se encuentra en la versión 1.1 (más adelante, en su sección correspondiente, se indicará el motivo de esto).

Por otro lado, cuando la aplicación realiza la transición de color la primera vez, se aprecia por unos pocos milisegundos el fondo de la pantalla en blanco. Esto se debe a que como el blanco es el color por defecto del fondo de las páginas y aplicaciones web, mientras se lee y procesa el *CSS* el navegador va a mostrar el fondo

en blanco, por lo que hasta que no se carguen éstas en el simulador no se muestran. Después, una vez se han seleccionado cada una de ellas, se realiza el cambio de manera correcta.

- *Simulador Firefox OS -App Manager-*

  Debido a que actualmente la versión que se emplea es la 26, como se acaba de comentar anteriormente, por una mayor seguridad, se decide emplear este otro simulador para verificar que la funcionalidad de la aplicación sigue siendo la esperada. Por lo tanto, para poder inicializarlo, se añade la aplicación (empaquetada en este caso, pues no se encuentra alojada en ningún servidor) y se selecciona la carpeta principal donde se encuentra ésta. Después, se procede a simularla:

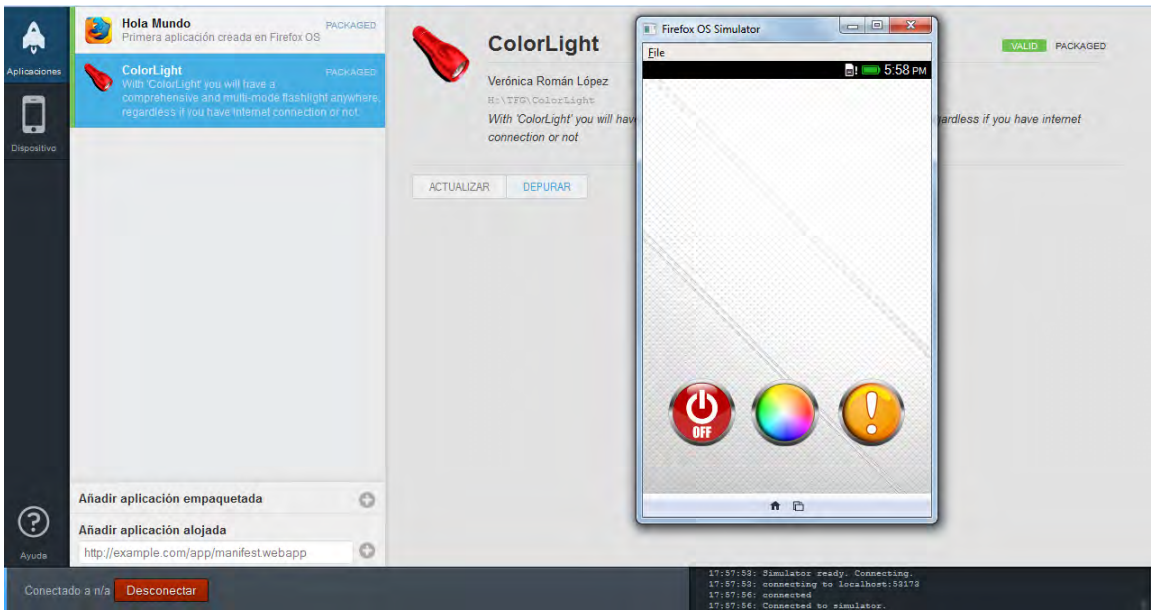


Figura 4.3: Emulando aplicación ColorLight al simulador App Manager

Cabe destacar, que en este caso, teniendo el dispositivo en versión *Firefox OS* 1.1 el simulador no lo reconoce, ya que como bien indica Mozilla, este simulador es apto para terminales que tengan una versión 1.2 o posterior. Pero por otro lado, hay que señalar que se puede simular correctamente la aplicación ya que la versión del navegador es la correcta, verificándose que ésta funciona igual que el anterior simulador.

También hay que indicar que en este simulador ocurre lo mismo que en el anterior en cuanto la aplicación realiza la transición de color la primera vez. Con el fin de que estos simuladores realizasen la transición correctamente, se procede a modificar el formato de las imágenes así como su tamaño para que sean lo menos pesadas posibles, pero aun así se sigue apreciando este detalle aunque sea por menos tiempo.

- *Simulador Ubuntu Touch OS*

  Para evaluar dicho estudio, en primer lugar se procede a comprobar si la funcionalidad de la aplicación que se ha desarrollado en *Firefox OS* es similar a la del sistema operativo *Ubuntu Touch OS*.

  En este caso, se a comprobado en la tabla 2.8 que los motores de navegación son similares, por lo que en un principio, la visualización debería ser semejante.

  A continuación, se visualiza la aplicación *ColorLight*, (creada desde *Firefox OS*), añadiendo los ficheros necesarios para que la aplicación se pueda ejecutar en *Ubuntu Touch OS* correctamente.

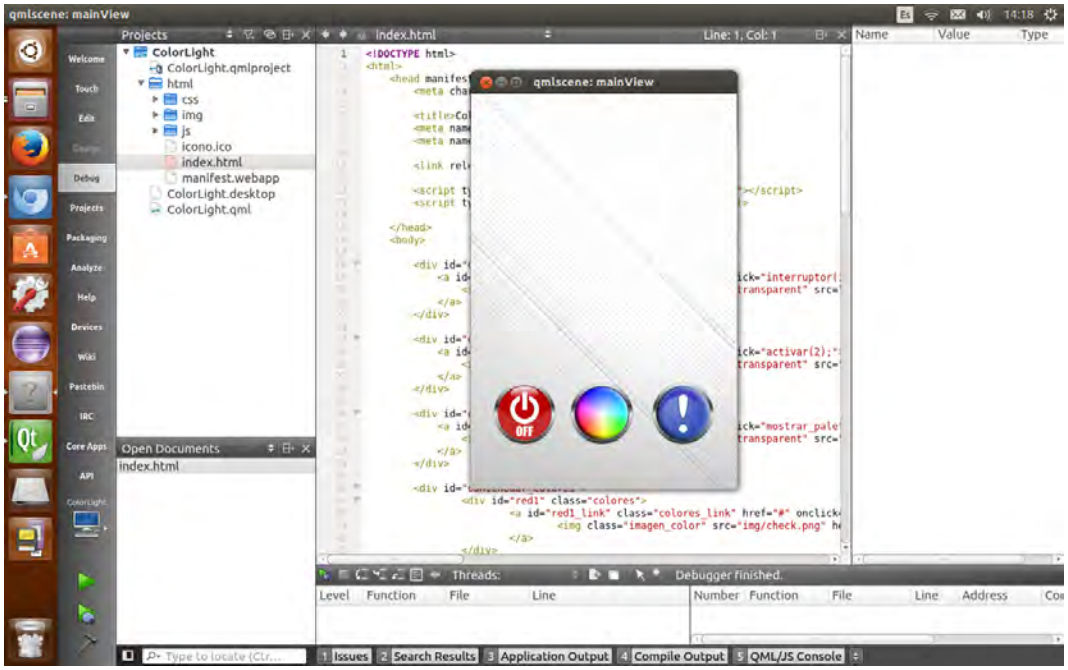


Figura 4.4: Aplicación ColorLight corriendo en Ubuntu Touch OS

NOTA: *Para obtener más información sobre los ficheros que hay que crear, véase el capítulo 3.1.*

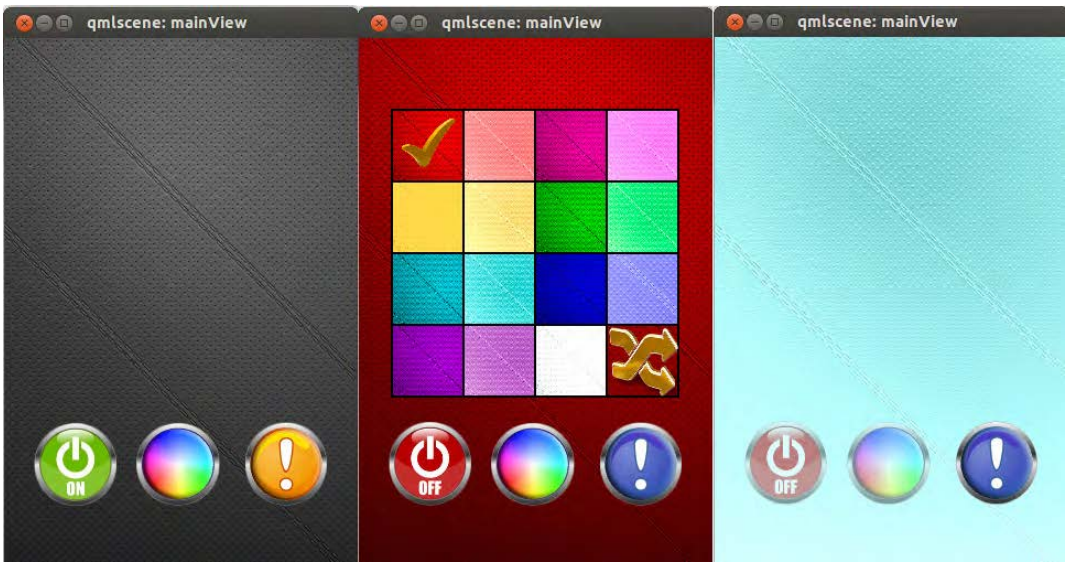


Figura 4.5: Aplicación ColorLight corriendo en Ubuntu Touch OS -continuación-

Finalmente, se corrobora que aunque el entorno de desarrollo sea diferente, al tener ambos (tanto *Firefox OS* como *Ubuntu Touch OS*) el mismo motor de navegador, *Gecko*, se comprueba que el modo de visualización sigue siendo el mismo que en el caso de los simuladores de *Firefox OS* y, del mismo modo, el funcionamiento de la aplicación es el correcto, ya que realiza las mismas acciones y de la misma manera que el sistema operativo en el que ha sido creada.

Aunque, cabe destacar, que el efecto de color que se encuentra en el archivo *CSS* (*inset*) y que se inserta en los bordes de la paleta de colores no es reconocido por *Ubuntu Touch OS*, al contrario que ocurre en el entorno de Mozilla. Y, además, en cuanto a la transición de color, ocurre algo parecido a los simuladores anteriores, y es que se aprecia también el fondo de la pantalla blanco, pero en este caso, se reproduce siempre que sucede el intercambio de colores.

- *Simulador Tizen OS*

  En este caso, dado que *Firefox OS* y *Tizen OS* tienen distinto motor de navegador (véase la tabla 2.8), se procede a comprobar si la funcionalidad de las dos aplicaciones que se han desarrollado en *Firefox OS* es similar a la del sistema operativo Tizen o no.

  A continuación, se muestra cómo se ejecuta la aplicación *ColorLight* en el emulador de *Tizen OS* tal cuál está programada para *Firefox OS*:

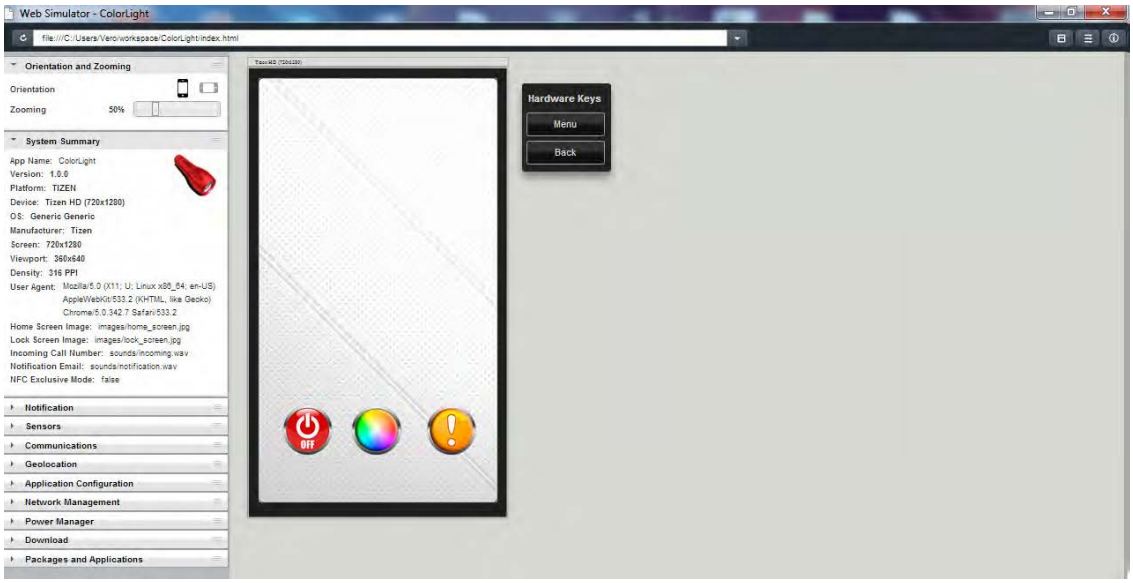


Figura 4.6: Aplicación ColorLight corriendo en Tizen

Figura 4.7: Aplicación ColorLight corriendo en Tizen -continuación-

En este entorno, dado que el motor del navegador de Tizen es *Webkit*, es decir, no es el mismo que emplea *Firefox OS* (*Gecko*), repercute, de algún modo, en la la visualización de la aplicación, ya que el navegador no lo interpreta de la misma manera que el de Mozilla. De ahí, que la paleta de colores no se vea en su cuadrante, ya que los establece en otra zona, aunque la funcionalidad de éstos siga estando en el mismo lugar. Además, al igual que ocurría en *Ubuntu Touch OS*, el efecto *inset* no es reconocido por Tizen tampoco, por lo que no hace nada con él y únicamente se visualiza en color negro. También hay que destacar que en este entorno se produce la misma limitación en cuanto a la transición de color que en el simulador anterior.

Por otro lado, cabe destacar que, independientemente de la visualización del cuadrante de la aplicación, el funcionamiento de ésta es el correcto, ya que realiza las mismas acciones y del mismo modo que el sistema operativo Firefox.

Pruebas realizadas en dispositivos reales

- *ZTE Open -Firefox OS-*

  Una vez se ha emulado la aplicación en ambos simuladores y ésta funciona correctamente, se procede a instalarla en el dispositivo de una de las dos maneras que ofrece *Firefox OS*.

  La primera de las opciones es la más común y es empaquetar el archivo, en este tipo de aplicación se encuentran todos los ficheros además del *manifest.webapp* y, además, de este modo se tiene acceso a todas las APIs. Y la segunda, alojar las aplicaciones en un servidor web (no es el caso). Este otro tipo de aplicación se denomina como hospedada. En este caso, los usuarios necesitan estar online para acceder a los recursos de la aplicación y los desarrolladores están limitados en el alcance de llamadas al *WebAPI*.

  Por lo que se escoge la primera de las opciones, y se instala directamente en el

dispositivo ZTE Open para ver si su funcionalidad es la correcta.

Figura 4.8: Aplicación ColorLight instalada en dispositivo ZTE OPEN

Figura 4.9: Aplicación ColorLight instalada en dispositivo ZTE OPEN -continuación-

Finalmente se comprueba que la funcionalidad de la aplicación es exactamente la misma que en los simuladores ejecutados en *Firefox OS*.

Hay que destacar que la versión del dispositivo sigue siendo 1.1 debido a que existen algunos problemas con el tema de actualizaciones entre Movistar y ZTE Open, por

lo que en estos momentos se hace imposible poder actualizar la versión a la actual 1.2 para comprobar si en el simulador *App Manager* es capaz de reconocer correctamente el dispositivo o no. En la siguiente imagen se muestra el error que muestra el dispositivo:

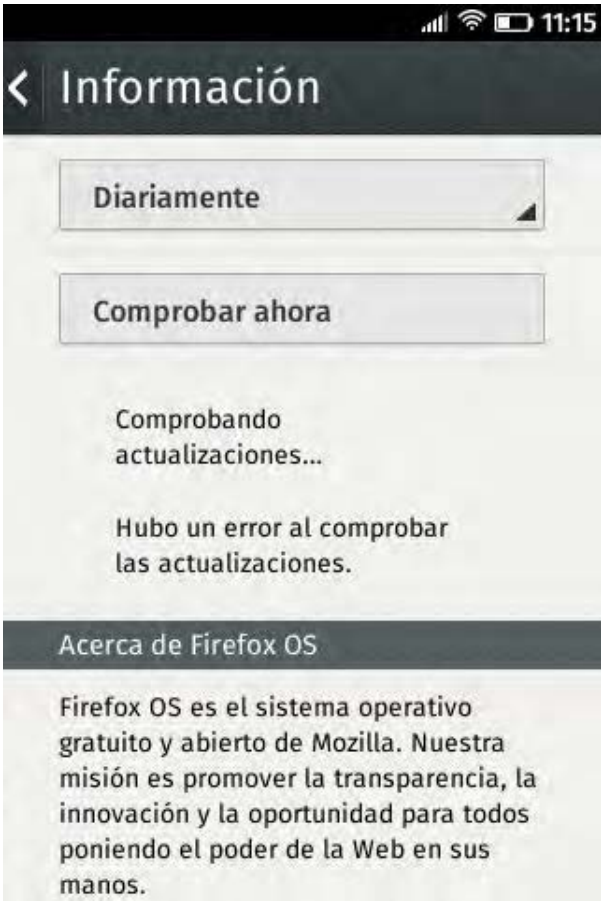


Figura 4.10: Error referente a las actualizaciones que se da en el dispositivo ZTE Open

- *THL W8s -Android*

  Por otro lado, para comprobar si esta aplicación funcionan en otro *smartphone* que no sea de *Firefox OS*, se decide ejecutarla en un dispositivo Android. Ya que las características de este sistema operativo son totalmente distintas a las de *Firefox OS*, y se realizan pruebas tanto en el navegador que trae por defecto el dispositivo, como en el navegador Firefox.

1. *Android Browser*

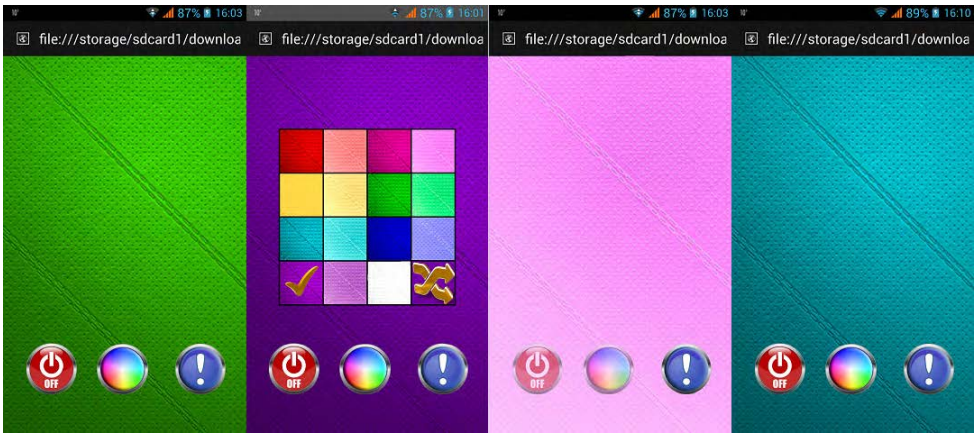


Figura 4.11: Aplicación ColorLight corriendo en el navegador de Android

2. *Navegador Firefox*

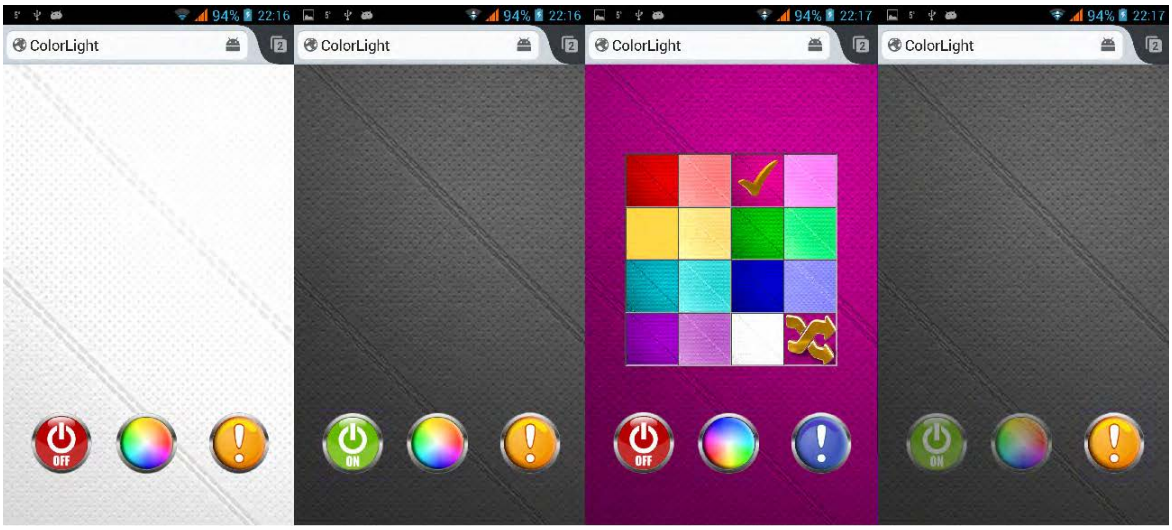


Figura 4.12: Aplicación ColorLight corriendo en el dispositivo Android con el navegador Firefox

Una vez realizadas las pruebas en el terminal Android, se comprueba que ambos navegadores el modo de visualización, así como el funcionamiento de la aplicación es el mismo que en *Firefox OS*. Salvo el efecto de *CSS* (*inset*), que el navegador nativo de Android no lo reconoce y, en cambio en el de Firefox sí, como era de esperar.

Destacar que también en ambos navegadores ocurre lo mismo en cuanto a la transición de los colores, la primera vez que procesa el cambio de imagen se muestra el fondo en blanco.

4.2.3. Images ScanApp

Al igual que ocurre con la creación de *ColorLight*, *Images ScanApp* ha sido simulada durante su desarrollo con el fin de que su funcionamiento fuera el esperado. Además, también se realizaron sobre ésta diversas pruebas en varios emuladores y dispositivos, ya que esta aplicación requiere de APIs pertenecientes a *Firefox OS* al contrario que *ColorLight*, que no depende de ningún sistema operativo móvil. Al finalizar dichas pruebas se extraerán una serie de conclusiones conjuntas.

Pruebas realizadas en emulación

- *Simulador Firefox OS -Complemento-*

  En primer lugar, se ejecuta la aplicación en este simulador:

Figura 4.13: A la izquierda se muestra la pantalla principal de la aplicación, en el centro y a la derecha se visualiza la información que proporciona el simulador cuando se pulsan los respectivos botones (Espacio Disponible - Espacio Ocupado).

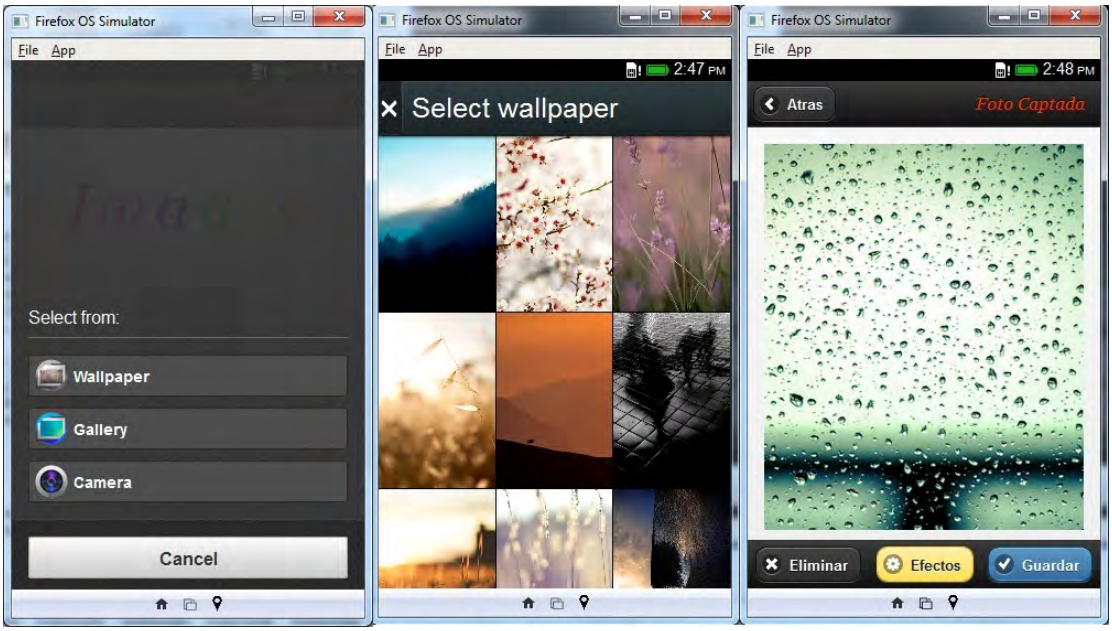


Figura 4.14: A la izquierda se muestra la Web Activity, en el centro aparece la lista de imágenes que se pueden visualizar como *Wallpapers*, y a la derecha se visualiza la imagen seleccionada, en la que se pueden añadir los efectos y demás.

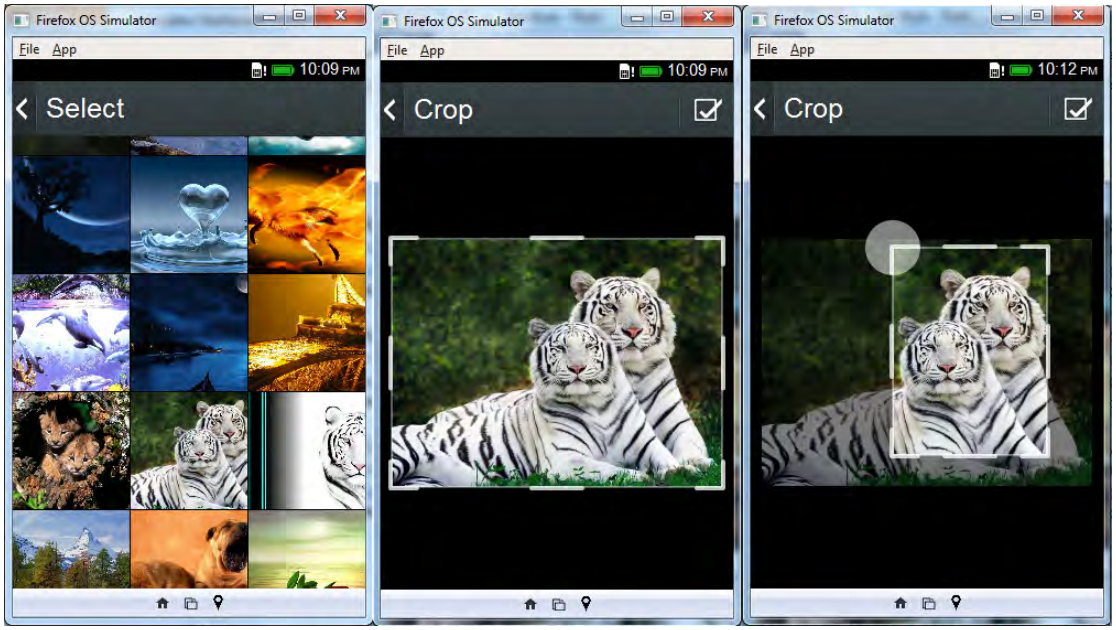


Figura 4.15: A la izquierda se muestran las imágenes de la galería, en el centro se visualiza la imagen seleccionada y a la derecha se muestra cómo se está seleccionando la parte que interesa conservar.

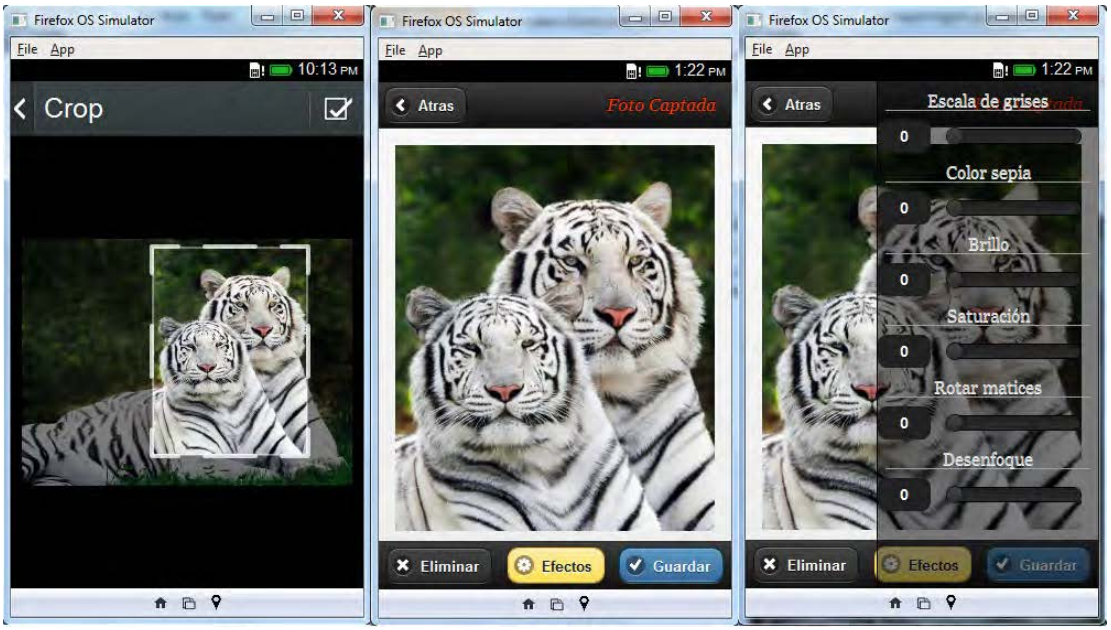


Figura 4.16: A la izquierda se muestra la imagen con la zona que se va a recortar, en el centro se visualiza la zona seleccionada en otra pantalla de la aplicación, y a la derecha se muestran los efectos disponibles.

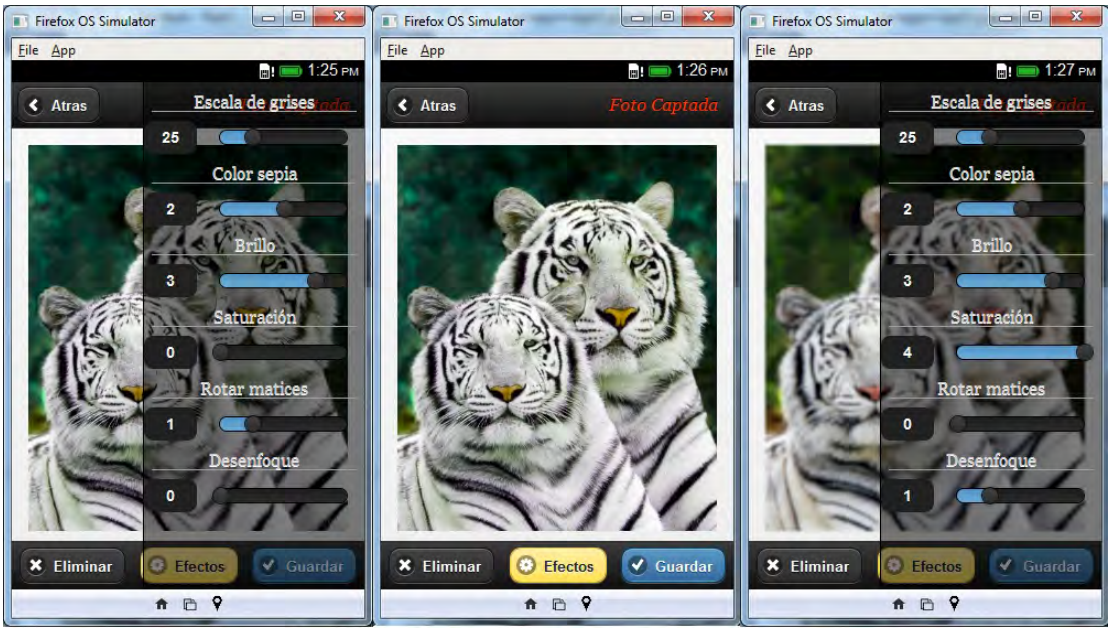


Figura 4.17: A la izquierda se muestra cómo están afectando los efectos sobre la imagen, en el centro se ve la imagen con los efectos aplicados y a la derecha aparece ésta de nuevo con otros efectos aplicados.

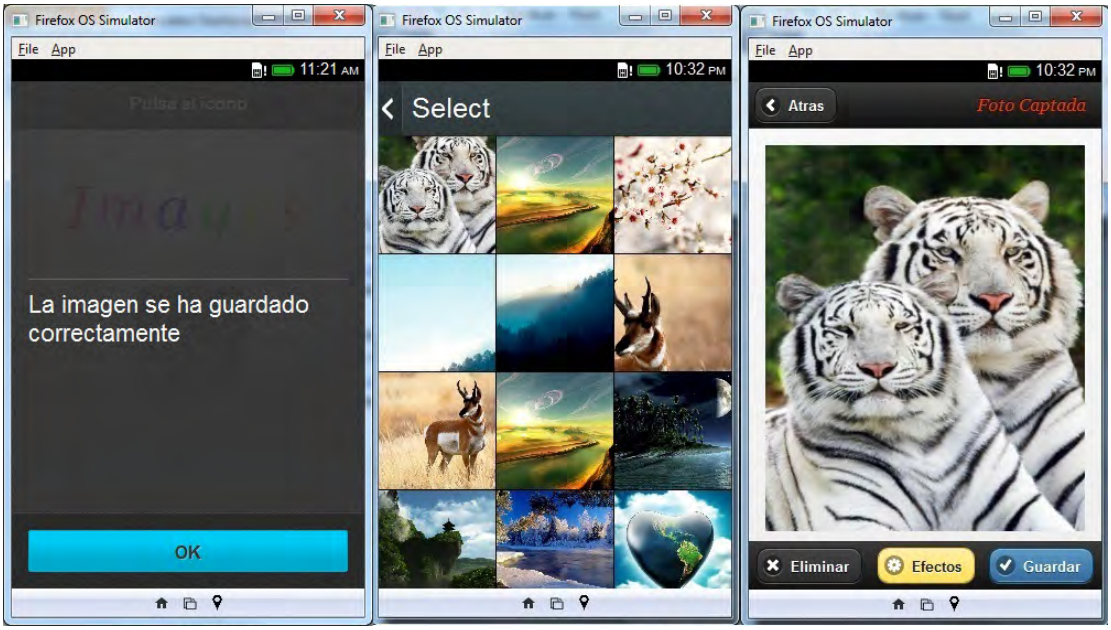


Figura 4.18: A la izquierda se muestra la pantalla cuando se ha guardado la imagen, en el centro se muestra el listado de la galería y se corrobora que se ha almacenado bien, y a la derecha se ve la imagen recortada, tal cual se almacenó.

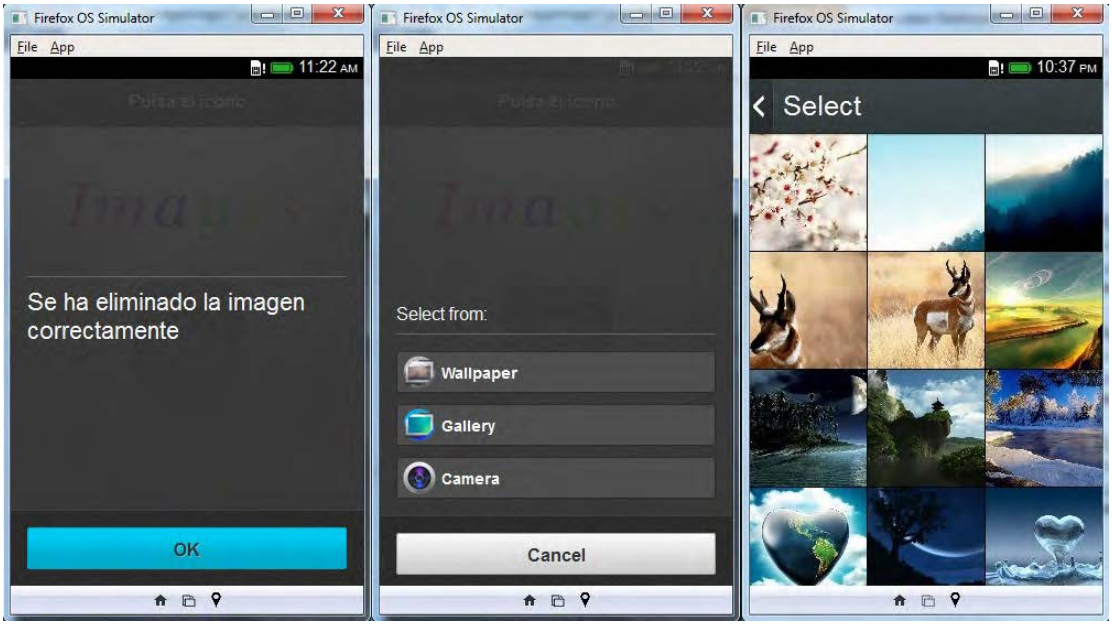


Figura 4.19: A la izquierda se visualiza la pantalla cuando se ha eliminado la imagen, en el centro se muestra la pantalla referente a la Web Activity y a la derecha se comprueba que la imagen ya no está almacenada en Galería.

Figura 4.20: Cámara bloqueada en Simulador Firefox OS -Complemento-

Viendo las figuras se puede comprobar que la aplicación funciona según lo esperado, salvo en el caso de la cámara que cuando se entra en ella, la aplicación se queda bloqueada. Se ha corroborado que éste no sea un error de la aplicación y se ha verificado que ésta funciona perfectamente en un dispositivo como se verá más adelante. El problema viene del propio simulador, corroborado por la gente de *Firefox OS*, quienes indican que la aplicación de la cámara todavía tiene algunos *bugs* y éstos aún no se han solventado.

- *Simulador Firefox OS -App Manager-*

  En el caso de éste simulador, se procede del mismo modo que en el anterior mostrando cómo se ha ejecutado la aplicación. Indicar que dicho simulador se diferencia del anterior porque en éste el botón inferior de geolocalización desaparece:

Figura 4.21: A la izquierda se muestra la pantalla principal de la aplicación, en el centro y a la derecha se visualiza la información que proporciona el simulador cuando se pulsan los respectivos botones (Espacio Disponible - Espacio Ocupado).

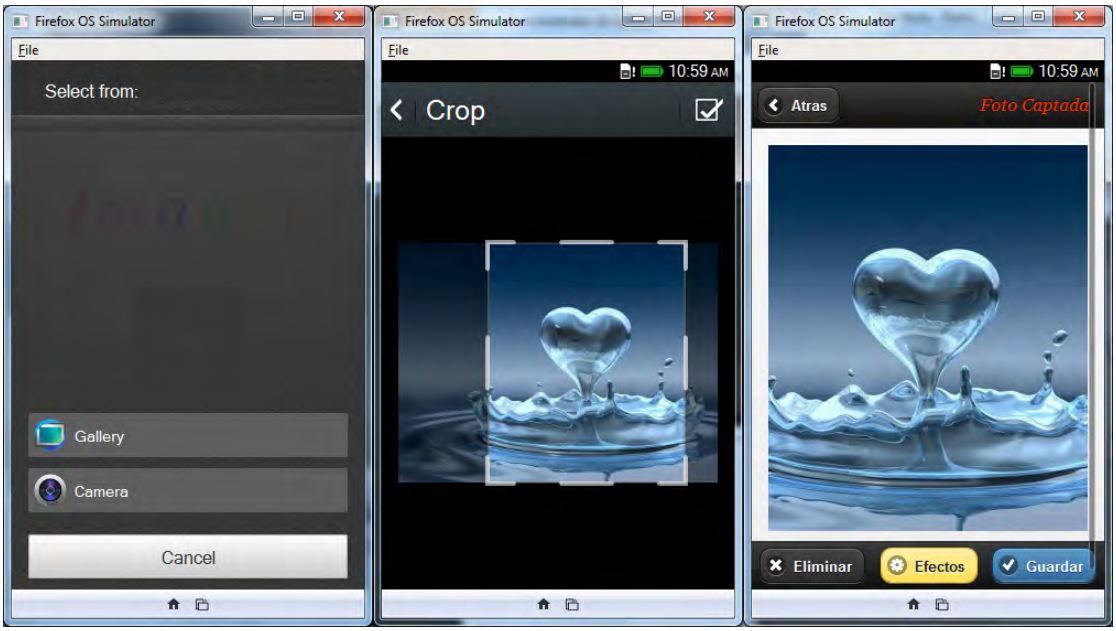


Figura 4.22: A la izquierda se muestra la Web Activity, en el centro aparece la imagen que va a ser recortada, y a la derecha se visualiza la imagen seleccionada y recortada, en la que se pueden añadir los efectos y demás.

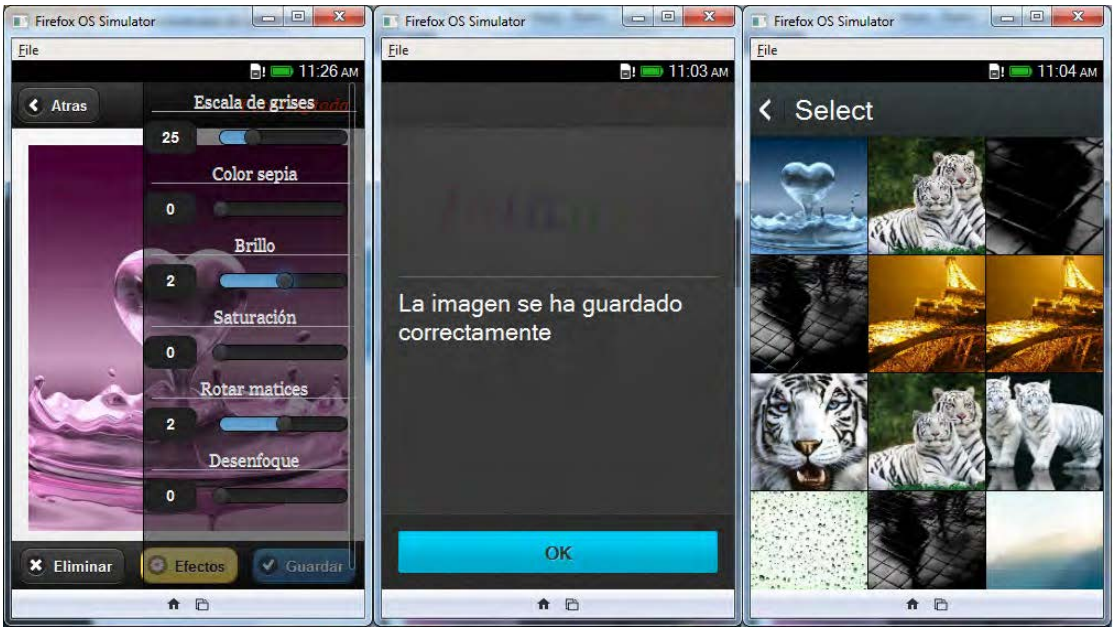


Figura 4.23: A la izquierda se muestra la imagen con los efectos aplicados, en el centro se almacena la imagen y a la derecha se comprueba que ésta se ha almacenado correctamente.

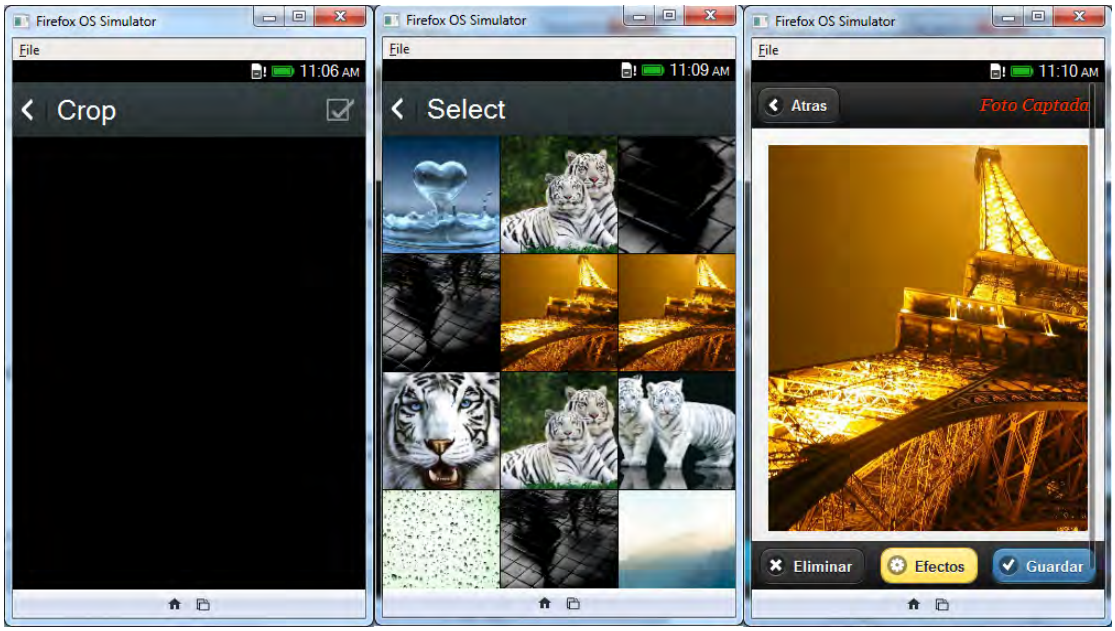


Figura 4.24: A la izquierda se muestra que el simulador no puede captar la imagen recortada, en el centro se comprueba que ésta existe y a la derecha se escoge una imagen para eliminarla.

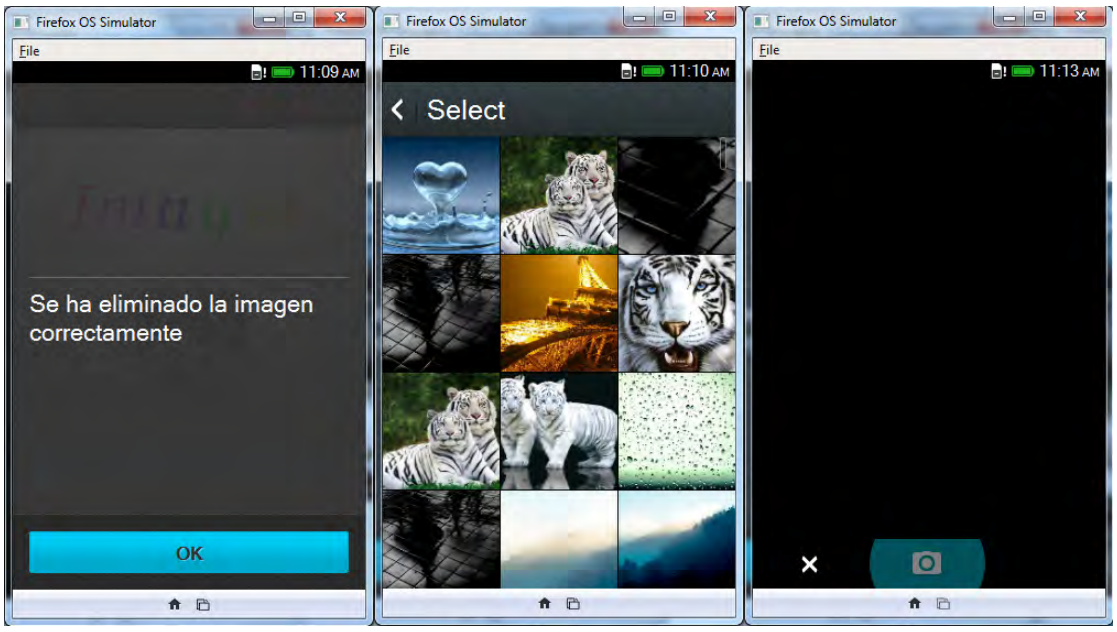


Figura 4.25: A la izquierda se muestra que la imagen se ha eliminado satisfactoriamente, en el centro se comprueba que la imagen se ha borrado correctamente y, finalmente, se observa cómo afecta el empleo de la cámara en el simulador, quedándose bloqueado.

En este simulador se han encontrado algunas limitaciones más que en el anterior. Aparte del bloqueo de la aplicación cuando entra en funcionamiento la cámara, éste no permite seleccionar de manera flexible la zona que se desea recortar en la imagen, al contrario que el anterior simulador. Además, al almacenarse una imagen recortada, cuando se accede de nuevo a ella ya no es capaz de mostrarla, en cambio en el otro simulador sí. Éste se debe a un error en el simulador, ya que más adelante se podrá comprobar que en el dispositivo funciona perfectamente, al igual que en el anterior simulador.

Por otro lado, cabe indicar que cuando se invoca a la *Web Activity*, ésta únicamente muestra las opciones de galería y cámara, no dando opción de elegir los *wallpapers*, cosa que no ocurre en el anterior simulador. Más adelante, se comprobará cómo lo muestra el dispositivo ZTE Open.

- *Simulador Ubuntu Touch OS*

  En este caso, al igual que sucede con *ColorLight*, la visualización de la aplicación debería ser la misma ya que *Ubuntu Touch OS* emplea el mismo motor de navegador que *Firefox OS*.

Figura 4.26: Aplicación Images ScanApp corriendo en Ubuntu Touch OS

Como se ha podido comprobar durante la simulación y, como estaba previsto, debido a que esta aplicación emplea APIs de *Firefox OS* como se comentó en capítulos posteriores no deja interactuar en ningún momento ya que *Ubuntu Touch OS* no sabe interpretar qué tiene que hacer cuando entran APIs ajenas a lo que conoce.

Cabe destacar que en este caso se muestra correctamente la primera pantalla debido a que el motor es el mismo que *Firefox OS* (*Gecko*) y ésta se basa en contenido *HTML*, *CSS* y *JavaScript*.

- *Simulador Tizen OS*

  Del mismo modo, se realizan las pruebas referentes a esta aplicación pero en este caso, simulándola desde *Tizen OS*:

Figura 4.27: Aplicación Images ScanApp corriendo en Tizen OS

Dado que este entorno emplea diferente motor de navegador con respecto a *Firefox OS*, se comprueba en esta aplicación que la visualización se ve afectada por este motivo. Además, al igual que ocurre en *Ubuntu Touch OS*, *Tizen* no sabe qué hacer cuando se interponen las APIs de *Firefox OS*. Por lo que la aplicación no realiza ninguna acción.

Pruebas realizadas en dispositivos reales

- *ZTE Open -Firefox OS-* Debido a que en los simuladores la cámara no funcionaba se comprobó si esto también sucedía en el dispositivo o no, a continuación se muestra la aplicación en plena ejecución:

Figura 4.28: Aplicación Images ScanApp instalada en dispositivo ZTE OPEN

Figura 4.29: Aplicación Images ScanApp instalada en dispositivo ZTE OPEN -continuación-

Figura 4.30: Imagen guardada en galería y eliminada en Images ScanApp

Figura 4.31: Realizando fotografía con Images ScanApp

Figura 4.32: Visualizando fotografía en galería de Images ScanApp y recortándola

De este modo se verifica que la aplicación funciona correctamente, visualizándose la cámara como es debido. Además, cuando se accede a la *Web Activity* ésta muestra las tres opciones al igual que el primer simulador de Firefox OS.

- *THL W8s -Android-*

  En este caso, se procede a comprobar el funcionamiento de esta aplicación, *Images ScanApp*, en ambos navegadores del dispositivo Android:

  1. *Android Browser*

Figura 4.33: Aplicación Images ScanApp corriendo en el navegador nativo de Android

  2. *Navegador Firefox*

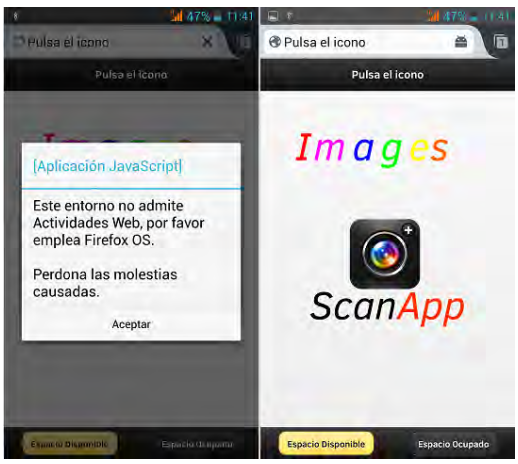


Figura 4.34: Aplicación Images ScanApp corriendo en el navegador Firefox Android

Se puede observar que en el caso del navegador Firefox, existe una ventana emergente en la que se especifica que no se puede ejecutar esta aplicación por el tema de

las *Web Activities*. Esto no se puede especificar en todos los navegadores, porque, por el momento, no hay una manera genérica de preguntar esto a todos, sino sólo a los de Firefox.

Por otro lado, al igual que ocurre en los otros simuladores, en ambos únicamente se puede ver la primera pantalla de la aplicación, ya que ésta está formada por *HTML*, *CSS* y *JavaScript* como se comentó anteriormente. Por lo que se comprueba que la interactuación es nula ya que Android no sabe interpretar las APIs del entorno de Mozilla.

Por otro lado, la visualización de la primera pantalla es prácticamente la misma que en *Firefox OS* dado que se emplea el mismo motor en el navegador, aunque luego no permita interactuar con ella.

A modo de **conclusión** y, después de los resultados obtenidos, cabe indicar que la primera aplicación, *ColorLight*, es totalmente versátil para cualquier entorno que se base en *HTML*, *CSS* y *JavaScript*, ya que, como se ha comprobado, aunque el motor del navegador sea distinto, no afecta al funcionamiento de ésta, sino sólo a la apariencia de la aplicación. Por lo que con pequeños cambios de estilo *CSS* **se puede portar cualquier aplicación, siempre y cuando ésta no incluya APIs nativas.** Además, al ser una aplicación web y, como se comentó anteriormente, **el rendimiento es menor que si la aplicación hubiera sido nativa**, por ello las imágenes tardan más en mostrarse.

Y eso es lo que ocurre con la segunda aplicación, *Images ScanApp*. Dado que ésta requiere acceso al *hardware* del dispositivo, se necesitan APIs nativas. Lo que acarrea un problema a la hora de traspasar la aplicación a otro sistema operativo móvil como se daba en los anteriores casos. Por lo que habría que realizar una aplicación distinta por cada sistema operativo móvil. Por lo que esta aplicación no es nada versátil.

Finalmente, a modo resumen se exponen ambas tablas en las que se muestran los resultados de las pruebas en todos los entornos, tanto emuladores como dispositivos reales:

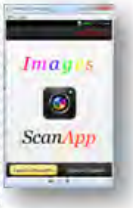
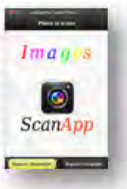
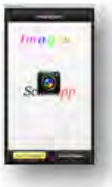
Simulador Firefox OS - Complemento - (Motor navegadores Gecko)		Simulador Firefox OS - App Manager - (Motor navegadores Gecko)		Ubuntu Touch (Motor navegador Gecko)		Tizen OS (Motor navegador Webkit)	
ColorLight	Images ScanApp	ColorLight	Images ScanApp	ColorLight	Images ScanApp	ColorLight	Images ScanApp
							

Tabla 4.1: Pruebas realizadas en los distintos emuladores

ZTE Open - Firefox OS - (Motor navegador Gecko)		THL W8s - Android - (Android Browser)		THL W8s - Android - (Navegador Firefox)	
ColorLight	Images ScanApp	ColorLight	Images ScanApp	ColorLight	Images ScanApp
					

Tabla 4.2: Pruebas realizadas en los dispositivos reales

Capítulo 5

Planificación y Presupuesto

5.1. Planificación

La planificación del presente proyecto se ha decidido realizar en horas, en previsión de que el tiempo al día dedicado a su realización no sería regular a lo largo del proceso de desarrollo, debido a la coincidencia de este proyecto con una beca de Prácticas en Empresa a lo largo de todo el periodo y con clases y prácticas de la Universidad, durante prácticamente la primera mitad del mismo. Dando lugar a dejar de lado el desarrollo del mismo una temporada, debido a la dedicación plena a actividades académicas. Esta pausa se muestra como una franja roja en la Figura 5.1.

Cabe indicar, que entre las reuniones presenciales con la directora del proyecto, se ha mantenido contacto vía correo electrónico para el seguimiento del mismo.

Otro diagrama, el de *Pert*, se adjunta en su correspondiente Anexo A.1 debido a su extensión. Dicho diagrama muestra las dependencias y la secuenciación de tareas, así como las tareas críticas que establecen el camino crítico que determina la duración mínima para realizar el presente proyecto.

Por otro lado, en la Tabla 5.1 se puede apreciar la Estructura de Descomposición de Trabajo (EDT) en la que se puede observar cuál es el desglose en tareas de cada una de

las fases generales que conforman la elaboración del presente proyecto y que se citan en el presupuesto, así como las horas planificadas a emplear en cada una de ellas.

	Nombre de tarea	Duración	Comienzo	Fin
1	**Trabajo Fin de Grado**	**492 horas**	**lun 12/11/12**	**mar 11/02/14**
2	**Documentación**	**45 horas**	**lun 12/11/12**	**lun 19/11/12**
3	Búsqueda de información de las diversas plataforn	45 horas	lun 12/11/12	lun 19/11/12
4	**Despliegue de los entornos de desarrollo**	**40 horas**	**lun 19/11/12**	**lun 02/12/13**
5	Preparación de las herramientas de trabajo	15 horas	lun 19/11/12	mié 21/11/12
6	Introducción a los SDK	25 horas	mié 21/11/12	lun 02/12/13
7	**Desarrollo de la primera aplicación en todos los**	**15 horas**	**lun 02/12/13**	**mié 04/12/13**
8	Búsqueda, realización de tutoriales...	15 horas	lun 02/12/13	mié 04/12/13
9	**Estudio y desarrollo de las aplicaciones en Firef**	**195 horas**	**mié 04/12/13**	**mié 08/01/14**
10	Creación de aplicación ColorLight	37 horas	mié 04/12/13	mié 11/12/13
11	Creación de aplicación ScanApp Images	130 horas	mié 11/12/13	jue 02/01/14
12	Corrección y depuración de ambas aplicaciones	28 horas	jue 02/01/14	mié 08/01/14
13	**Evaluación del estudio**	**52 horas**	**mié 08/01/14**	**jue 16/01/14**
14	Pruebas de entorno en los diferentes sistemas ope	20 horas	mié 08/01/14	vie 10/01/14
15	Pruebas en dispositivo ZTE Open (Firefox OS)	5 horas	vie 10/01/14	lun 13/01/14
16	Pruebas en dispositivo THL W8s (Android)	2 horas	lun 13/01/14	lun 13/01/14
17	Evaluación de las pruebas realizadas	25 horas	lun 13/01/14	jue 16/01/14
18	**Redacción de la memoria**	**145 horas**	**jue 16/01/14**	**mar 11/02/14**
19	Redacción de la memoria	130 horas	jue 16/01/14	vie 07/02/14
20	Corrección y maquetación	15 horas	vie 07/02/14	mar 11/02/14

Tabla 5.1: Tabla EDT del proyecto

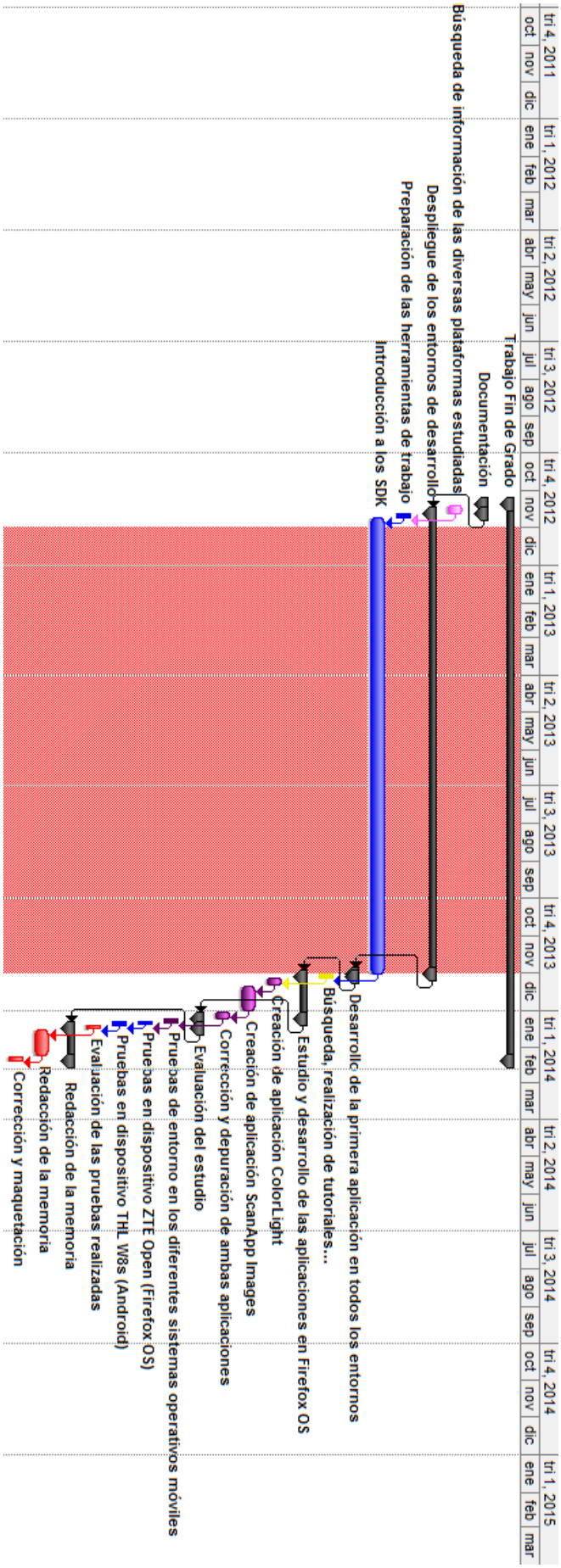


Figura 5.1: Diagrama de Gantt del proyecto

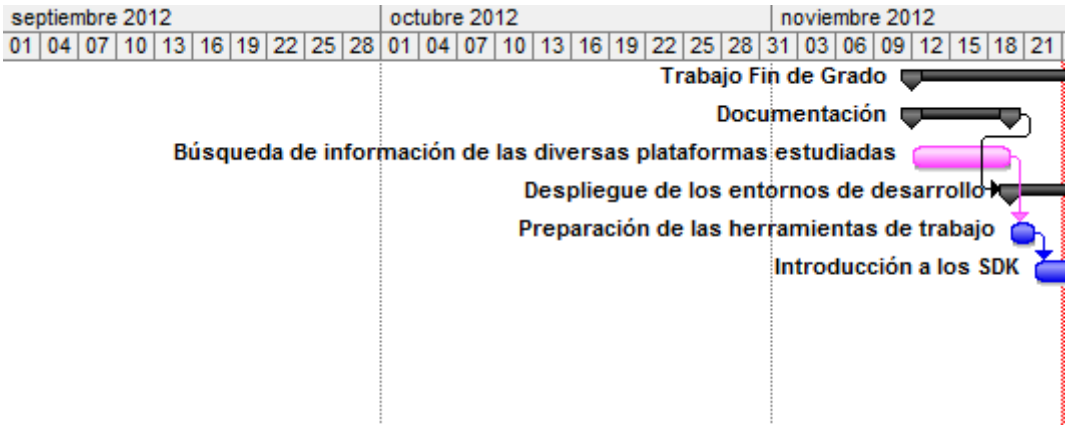


Figura 5.2: Vista ampliada del Diagrama de Gantt - Parte de documentación y despliegue

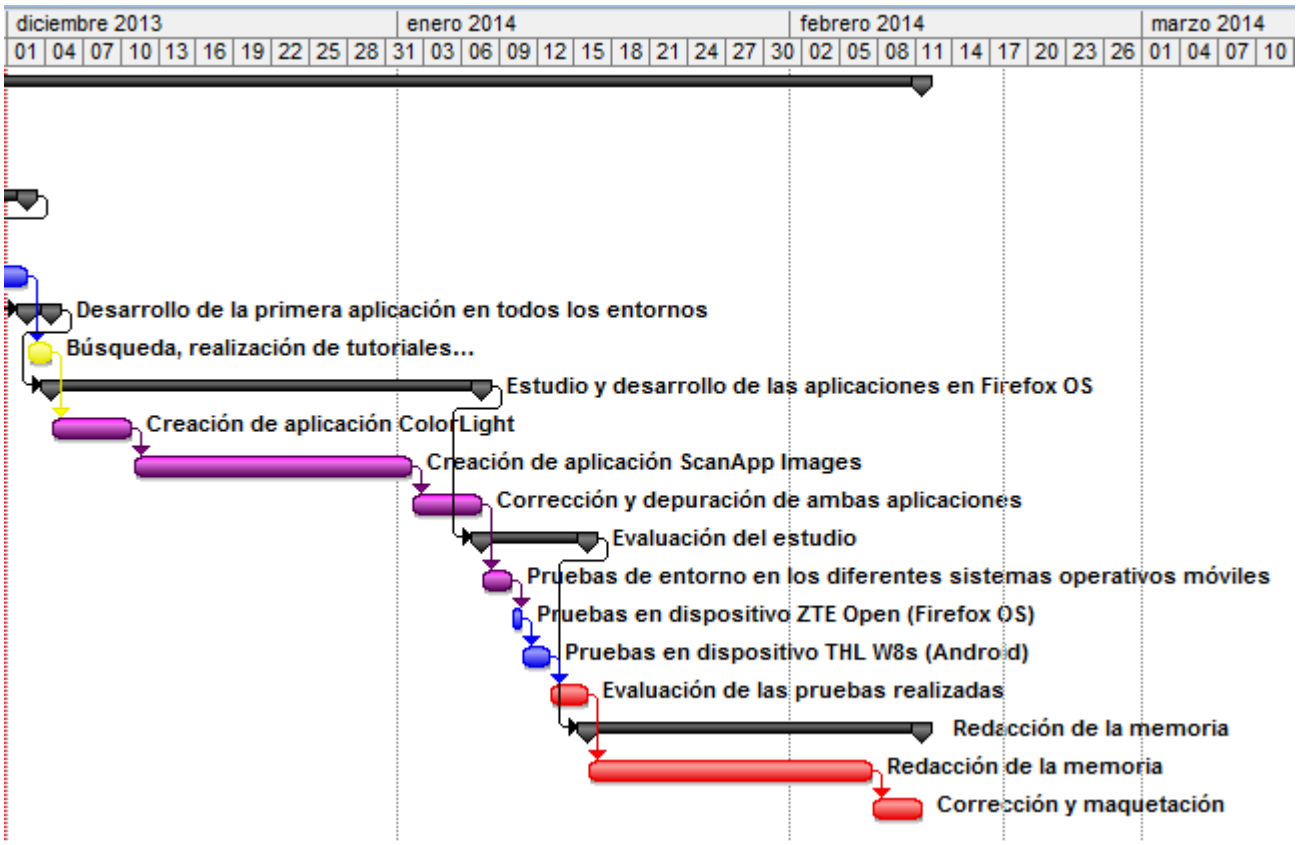


Figura 5.3: Vista ampliada del Diagrama de Gantt - Parte de desarrollo y evaluación

5.2. Presupuesto

En este capítulo, se lleva a cabo un desglose de las tareas realizadas durante el presente Trabajo Fin de Grado, lo que facilitará posteriormente el cálculo aproximado sobre su coste global. Tales costes, imputables a gastos de personal y a materiales empleados, se pueden deducir de las Tablas 5.2 y 5.3.

Debido a que este estudio ha estado marcado por diferentes fases o etapas como se comentó en el capítulo 1.1 de la presente memoria, a continuación se muestra la distribución de horas empleadas en cada una de dichas fases:

Fase 1: Documentación

1. Búsqueda de información de las diversas plataformas estudiadas (*WebOS*, *Firefox OS*, *Tizen OS*, *Ubuntu Touch OS* y *Sailfish OS*) (45 horas)

Fase 2: Despliegue de los entornos de desarrollo

1. Preparación de las herramientas de trabajo (Instalación de Sistema Operativo Ubuntu, Maquina Virtual -Virtual Box-, software requerido para la creación del estudio así como de la memoria, etc) (15 horas)
2. Introducción a los SDKs y familiarización de las APIs(25 horas)

Fase 3: Desarrollo de la primera aplicación en todos los entornos

1. Búsqueda, realización de tutoriales y aplicaciones sencillas para todas las plataformas estudiadas (15 horas)

Fase 4: Desarrollo de las aplicaciones en Firefox OS

1. Creación de aplicación *ColorLight* (37 horas)
2. Creación de aplicación Images ScanApp (130 horas)
3. Corrección y depuración (*ColorLight* + *Images ScanApp*) (28 horas)

Fase 5: Estudio y Evaluación del estudio

1. Pruebas de entorno en los diferentes sistemas operativos móviles (Simuladores) (20 horas)
2. Pruebas en dispositivo ZTE Open (*Firefox OS*) (5 horas)
3. Pruebas en dispositivo THL W8s (Android) (2 horas)
4. Evaluación de las pruebas realizadas (25 horas)

Fase 6: Redacción de la memoria

1. Redacción de la memoria (130 horas)
2. Corrección y maquetación (15 horas)

5.2.1. Costes de Personal

Para poder establecer el presupuesto asociado a los honorarios correspondientes a las personas que se encuentran a cargo de este proyecto, es necesario tener en cuenta las horas de trabajo realizadas en el mismo. Para ello, en la Tabla 5.2 se muestran las fases del proyecto y el tiempo aproximado que se ha dedicado a cada una de ellas. Así pues, se desprende que el tiempo total dedicado a dicho proyecto ha sido de 492 horas, de las cuales aproximadamente unas 20 horas han sido compartidas con la directora del proyecto, por lo que el total asciende a **512 horas**. Teniendo en cuenta que la tabla de honorarios del Colegio Oficial de Ingenieros Técnicos de Telecomunicación establece unas tarifas de 35,00 €/hora, el coste de personal se sitúa en **17.920,00 €**.

5.2.2. Costes de Material

En la Tabla 5.3 se desglosan los costes de los materiales empleados. En ella se ha considerado que el periodo de amortización sea de 1/4 según la vida media de cada elemento que así lo requiera, ya que algunos de estos materiales podrán ser reutilizados tras la realización de este proyecto.

Tabla 5.2: *Desglose de tareas y horas invertidas*

Fases	Descripción	Horas
Fase 1	*Documentación*	45 *horas*
Fase 2	*Despliegue de los entornos de desarrollo*	40 *horas*
Fase 3	*Desarrollo de la primera aplicación en todos los entornos*	15 *horas*
Fase 4	*Desarrollo de las aplicaciones en Firefox OS*	195 *horas*
Fase 5	*Estudio y Evaluación del estudio*	52 *horas*
Fase 6	*Redacción de la memoria*	145 *horas*
TOTAL		**492 *horas***

Tabla 5.3: *Costes de Material*

Concepto	Precio	Amortización	Importe
Dispositivo ZTE Open	80,00 €	1/4	20,00 €
Dispositivo THL W8s	265,50 €	1/4	66,38 €
Ordenador portátil HP Pavilion dv5	856,84 €	1/4	214,21 €
Ordenador de mesa Packard Bell Imedia D5000)	1184,13 €	1/4	296,03 €
S.O. Linux Ubuntu 12.04	0,00 €		0,00 €
S.O. Windows 7 Ultimate	0,00 €		0,00 €
Virtual Box v4.2	0,00 €		0,00 €
SDKs	0,00 €		0,00 €
Eclipse IDE para desarrolladores JavaScript Web	0,00 €		0,00 €
Tizen IDE	0,00 €		0,00 €
Qt Creator	0,00 €		0,00 €
Sailfish IDE	0,00 €		0,00 €
Adobe Dreamweaver CS6	291,85 €	1/4	72,96 €
Adobe Photoshop CS6	942,82 €	1/4	235,71 €
TeXnicCenter	0,00 €		0,00 €
Adobe Reader XI	0,00 €		0,00 €
Microsoft Project Professional 2007	995,95 €	1/4	248,99 €
Gastos transporte e Internet	120,00 €		120,00 €
TOTAL			**1.274,28 €**

5.2.3. Costes Totales

Tras haber establecido los costes personales y materiales asociados al proyecto, se puede proporcionar un presupuesto total, el cuál se muestra en la Tabla 5.4.

Tabla 5.4: *Presupuesto*

Concepto	**Importe**
Costes de Personal	17.920,00 €
Costes Materiales	1.274,28 €
Costes Indirectos (20 %)	3.838,86 €
Base imponible	23.033,14 €
I.V.A. (21 %)	4.836,96 €
TOTAL	**27.870,10** €

Por lo que, el presupuesto total del Trabajo Fin de Grado asciende a un total de **VEINTISIETE MIL OCHOCIENTOS SETENTA CON DIEZ CÉNTIMOS.**

Capítulo 6

Futuras líneas para ampliar el proyecto y Conclusiones

6.1. Futuras líneas para ampliar el proyecto

Aunque los requisitos del presente proyecto se han cumplido, estos siempre se pueden ampliar y mejorar en un futuro. Por este motivo, a continuación se proponen los trabajos futuros que se pueden realizar, manteniendo un orden de prioridad:

6.1.1. Realizar aplicación referente al audio

Con el fin de que el estudio sea más completo y se pueda analizar todo lo relacionado con el área multimedia, se podría evaluar cómo afectan las APIs referentes al audio con respecto a los otros sistemas operativos.

6.1.2. Tratamiento implícito de las imágenes en Images ScanApp

También sería buena idea que las imágenes se pudieran almacenar con los efectos. Para ello, habría que evaluar cómo se podrían modificar los píxeles en sí para que se puedan ajustar los filtros más comunes.

6.1.3. ColorLight con "flash"

Dado que en esta aplicación no se quería incluir ningún tipo de API nativa para no desviar el estudio, y, además, el primer dispositivo que *Firefox OS* ha sacado al mercado no dispone de *flash*, se deja como trabajo futuro, que cuando el API de *HTML 5* incluya el acceso al mismo, se proceda a introducir ese API en la aplicación y evaluarla.

6.2. Conclusiones

Después de haber realizado las pruebas convenientes en dicho estudio, se toma como conclusión que **la aplicación web que se desarrolle no debe de contener APIs nativas, para que sea totalmente versátil en cualquiera de los sistemas operativos móviles estudiados**, es decir, no porque sea una aplicación web significa que sea versátil. Con este fin fue creada la aplicación *ColorLight*. El problema que tienen este tipo de aplicaciones, es que no pueden tener acceso al *hardware* del dispositivo en ningún momento, por lo que su funcionamiento es más básico.

En el caso de *Images ScanApp* ocurre lo contrario. Dado que esta aplicación sí necesita tener acceso al dispositivo, requiere emplear APIs nativas. Esto conlleva ciertos límites, y es que en el momento que la aplicación se cambie de entorno, éste no sabrá interpretar lo que le indiquen esas APIs. Por lo que de este modo obliga a tener que diseñar una aplicación distinta para cada sistema operativo móvil. Corroborando de este modo que esta aplicación web no es multiplataforma como la anterior.

Por este motivo, aunque actualmente el API de *HTML 5* ofrece cierta funcionalidad interoperable sobre el acceso al *hardware* en cuanto al almacenamiento, geolocalización, vibración del dispositivo se refiere, aún no contempla algunas APIs referentes al entorno multimedia, como la cámara y el *flash*.

6.2.1. Consecución de objetivos

En el inicio del proyecto se estableció un objetivo principal además de una serie de objetivos específicos cuya consecución determinaría el resultado del presente Trabajo Fin de Grado:

- Se ha comparado el desarrollo web frente al desarrollo con otros lenguajes (o nativos).
- Se ha estudiado y comprendido cómo es la arquitectura y el funcionamiento de los sistemas operativos basados en Web para dispositivos móviles.
- Se han estudiado los entornos de desarrollo de las diversas plataformas, consiguiendo así tener una visión global de las características y funcionalidades que proporcionan cada una de ellas.
- Se ha aprendido a manejar los entornos de desarrollo de las plataformas estudiadas, lo que ha permitido realizar diversas aplicaciones y las pruebas necesarias sobre los simuladores y los dispositivos físicos.

Gracias a la consecución de todos y cada uno de los objetivos parciales establecidos, se ha conseguido alcanzar el objetivo principal del susodicho estudio. Se ha logrado implementar con éxito dos aplicaciones en entorno *Firefox OS*, y se han portado tanto al sistema que más características posee en común con él (*Ubuntu Touch OS*), como con el que menos (*Tizen OS*), evaluándose, de este modo, las APIs multimedia que se emplean en dichas aplicaciones. Además, ambas se han ejecutado en el dispositivo ZTE Open (*Firefox OS*) para comprobar su correcto funcionamiento, así como en un *smartphone* con sistema operativo Android.

Finalmente, a partir de las pruebas realizadas se ha llegado a una serie de conclusiones, pudiendo terminar de manera satisfactoria el presente estudio.

6.2.2. Reflexiones personales

Para la realización de este Trabajo Fin de Grado ha sido necesario el estudio de varias tecnologías y plataformas de las cuales no se tenía mucha noción, lo cual ha supuesto un esfuerzo personal para adquirir los conocimientos necesarios para entender cada una de ellas.

Además, al tratarse de sistemas operativos tan novedosos, surgieron algunos imprevistos, viéndose afectado el presente estudio, el cuál tuvo que ser encaminado hacia otra dirección en varias ocasiones. Esto conllevó a variar las estimaciones del tiempo que sería necesario invertir para realizar el presente Trabajo Fin de Grado, sumándose a esto el trabajo y las clases de la Universidad.

Asimismo, la falta de documentación que existe sobre estas plataformas, ha hecho necesario establecer contacto con otros desarrolladores a través de foros y demás, proporcionando una experiencia muy enriquecedora y útil. Sumado al hecho de tener que leer toda la documentación en inglés y la necesidad de expresarse en este idioma para plantear las dudas a los desarrolladores, ha supuesto una oportunidad para practicar con esa lengua. A pesar de ello, la falta de documentación que existe en estos momentos, ha sido la parte más negativa, ya que establece una gran barrera de entrada y dificulta la programación.

A modo de conclusión, el hecho de ver concluido el presente estudio, y obteniendo los resultados que se esperaban después de todo el tiempo y esfuerzo empleado; produce una grata satisfacción y hace tener la sensación de que haya merecido la pena tanta dedicación y esfuerzo.

Bibliografía

[1] A. H. Escobar, "Dispositivos moviles." Website, 2014. http://www.buenastareas.com/ensayos/Dispositivos-Moviles/3074241.html.

[2] T. Rodriguez, "El futuro del desarrollo movil." Website, 2014. http://www.genbetadev.com/desarrollo-aplicaciones-moviles/el-futuro-del-desarrollo-movil-crear-una-web-movil-o-una-aplicacion-nativa.

[3] CursosTech, "El futuro del desarrollo movil." Website, 2014. http://cursostech.pe/node/51.

[4] Baudlos, "Desarrollo de aplicaciones moviles." Website, 2013. http://baud1os.wordpress.com/.

[5] V. Lucian, "Desarrollo de una aplicacion hibrida para smartphone." Website, 2013. http://academica-e.unavarra.es/bitstream/handle/2454/7544/578081.pdf.

[6] M. David, *HTML5 Mobile Websites*. Focal Press, 2013.

[7] S. Chuan, *HTML5 Mobile Development Cookbook*. Packt Publishing, 2012.

[8] E. Weyl, *Mobile HTML5*. O'Reilly Media, Inc., 2013.

[9] J. Franganillo, "Html5: el nuevo estandar basico de la web." Website, 2014. http://www.academia.edu/2986830/HTML5_el_nuevo_estandar_basico_de_la_web.

[10] A. Kosmaczewski, *Mobile JavaScript Application Development*. O'Reilly Media, Inc., 2013.

[11] D. F. Pueyo, "Analisis y desarrollo de una aplicacion web para la publicacion y venta de productos." Website, 2013. http://zaguan.unizar.es/TAZ/EINA/2012/8001/TAZ-PFC-2012-389.pdf.

[12] S. Ausere, "Recomendaciones de frameworks html5 css3." Website, 2013. http://webysocialmedia.es/2012/recomendaciones-de-frameworks-html5-css3/.

[13] Wikipedia, "jquery." Website, 2014. http://es.wikipedia.org/wiki/JQuery#cite_note-1.

[14] jQuery Foundation, "jquery mobile." Website, 2014. http://jquerymobile.com/.

[15] W3Techs, "Uso de bibliotecas de javascript." Website, 2014. http://w3techs.com/technologies/overview/javascript_library/all.

[16] jQuery Foundation, "Licencias jquery foundation." Website, 2014. https://jquery.org/license/.

[17] M. D. N. y colaboradores individuales, "Mozilla developer network." Website, 2013. https://developer.mozilla.org/es/.

[18] G. E. Linares, "Nuevos sistemas operativos para celulares." Website, 2013. http://opinadel1al10.blogspot.com.es/2013/09/nuevos-sistemas-operativos-para.html.

[19] MozillaWiki, "B2g - architecture." Website, 2013. https://wiki.mozilla.org/B2G/Architecture.

[20] W. Foundation, "Firefox os." Website, 2013. http://es.wikipedia.org/wiki/Firefox_OS.

[21] M. D. N. y los contribuyentes individuales, "Web api." Website, 2013. https://developer.mozilla.org/en-US/docs/WebAPI/.

[22] a. L. F. P. Tizen Project, "Tizen developers." Website, 2014. https://developer.tizen.org/es.

[23] W. Foundation, "Tizen." Website, 2013. http://es.wikipedia.org/wiki/Tizen.

[24] C. C. A. 3.0, "Tizen web device referencia api." Website, 2014. https://developer.tizen.org/dev-guide/2.2.1/org.tizen.web.device.apireference/index.html.

[25] L. Marquez, "Todo sobre el nuevo sistema operativo tizen os." Website, 2013. http://luismarquez1965.blogspot.com.es/2013/11/todo-sobre-el-nuevo-sistema-operativo.html.

[26] S. M. Solis, "Como ir desarrollando en tizen os, tizenspain." Website, 2013. http://tizenspain.com/page/2/.

[27] C. L. Ubuntu, "Ubuntu developers." Website, 2013. http://developer.ubuntu.com/.

[28] F. Eroski, "Nuevos sistemas operativos para moviles." Website, 2013. http://www.consumer.es/web/es/tecnologia/software/2013/04/11/216372.php.

[29] C. L. Ubuntu and Canonical, "Ubuntu api." Website, 2014. http://developer.ubuntu.com/api/.

[30] Wikipedia, "Phonegap." Website, 2014. http://es.wikipedia.org/wiki/PhoneGap.

[31] S. Project, "Sailfish os." Website, 2014. https://sailfishos.org/.

[32] C. Valero, "Los nuevos sistemas operativos moviles que llegaran en 2013." Website, 2013. http://www.adslzone.net/article11636-los-nuevos-sistemas-operativos-moviles-que-llegaran-en-2013.html.

[33] M. Januszewski, "Api sailfish." Website, 2014,. http://sailfish.us.edu.pl/api.html.

[34] L. Electronics, "Open webos." Website, 2013. http://www.openwebosproject.org/.

[35] WordPress, "Lo mejor de lg en el mobile world congress 2013." Website, 2013. http://tecnologiaynoticias.info/2013/03/lo-mejor-de-lg-en-el-mobile-world-congress-2013/.

[36] L. Hewlett-Packard Development Company, "Webos aplication apis." Website, 2013. http://https://developer.palm.com/content/api/reference/application-apis.html.

[37] a. L. F. P. Tizen Project, "Guia de api tizen." Website, 2014,. https://developer.tizen.org/es/documentation/articles/system-information-api-guide.

[38] H.-P. D. Company, "Documentation webos." Website, 2013. https://developer.palm.com/content/api/index.html.

[39] T. P. a Linux Foundation Project, "Instalacion del sdk de tizen." Website, 2013. https://developer.tizen.org/es/downloads/sdk/installing-tizen-sdk?langredirect=1.

[40] S. OS, "Instalacion del sdk de sailfish." Website, 2013. https://sailfishos.org/develop-installation-article.html.

[41] C. L. U. y Canonical, "Instalacion del sdk de ubuntu." Website, 2013. http://developer.ubuntu.com/apps/create/get-the-sdk/.

ANEXOS

ANEXO A

Diagrama PERT

A.1. Diagrama PERT

A continuación, en las próximas páginas pertenecientes a este Anexo se mostrará el *Diagrama de Pert* referente al presente proyecto. Cabe destacar que éste se ha tenido que distribuir en varias páginas debido a sus dimensiones.

NOTA: *Indicar que para la realización de estos diagramas (Pert y Gantt), y la Tabla EDT se ha decidido emplear el software Microsoft Project Professinal 2007.*

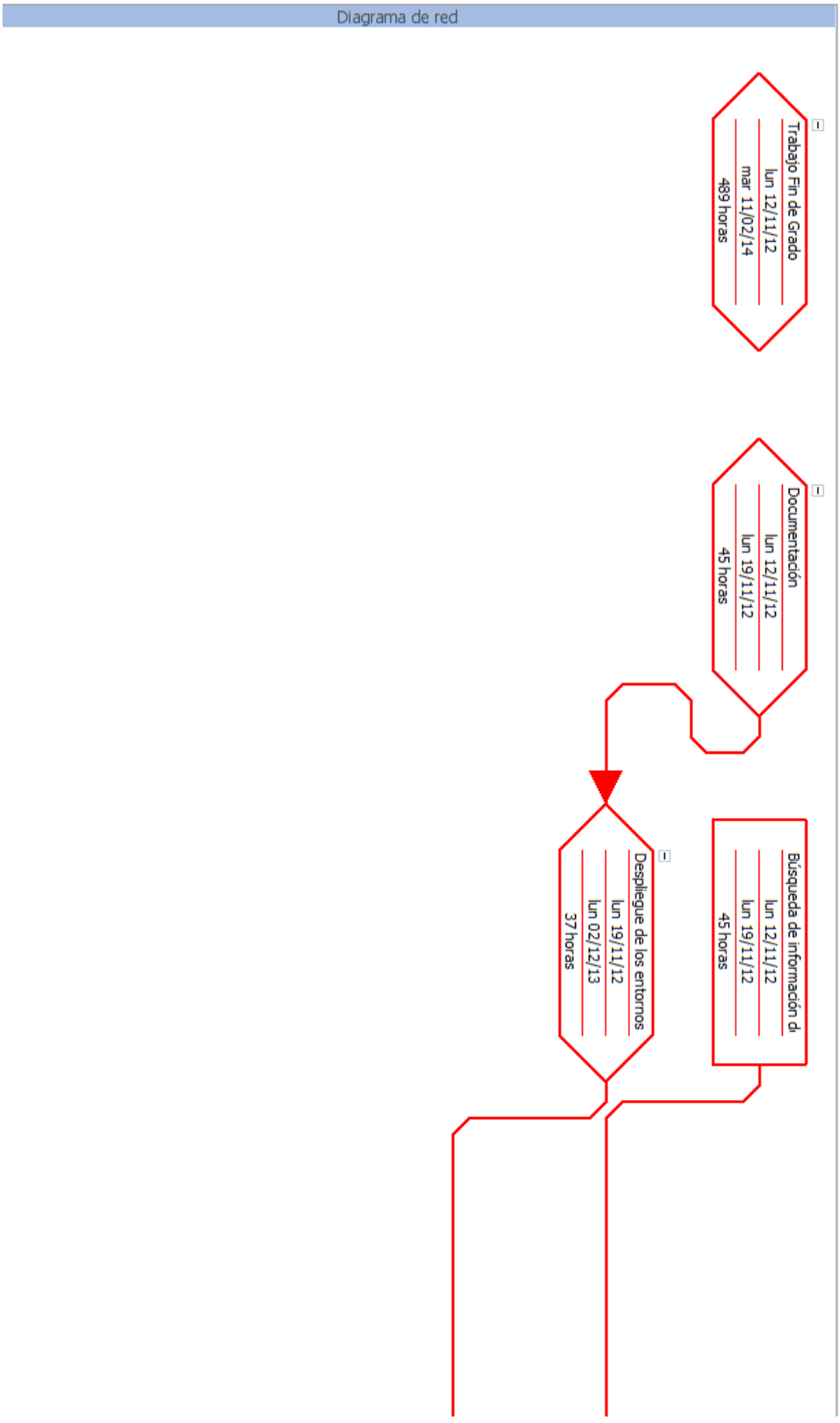


Figura A.1: Diagrama de Pert del Proyecto

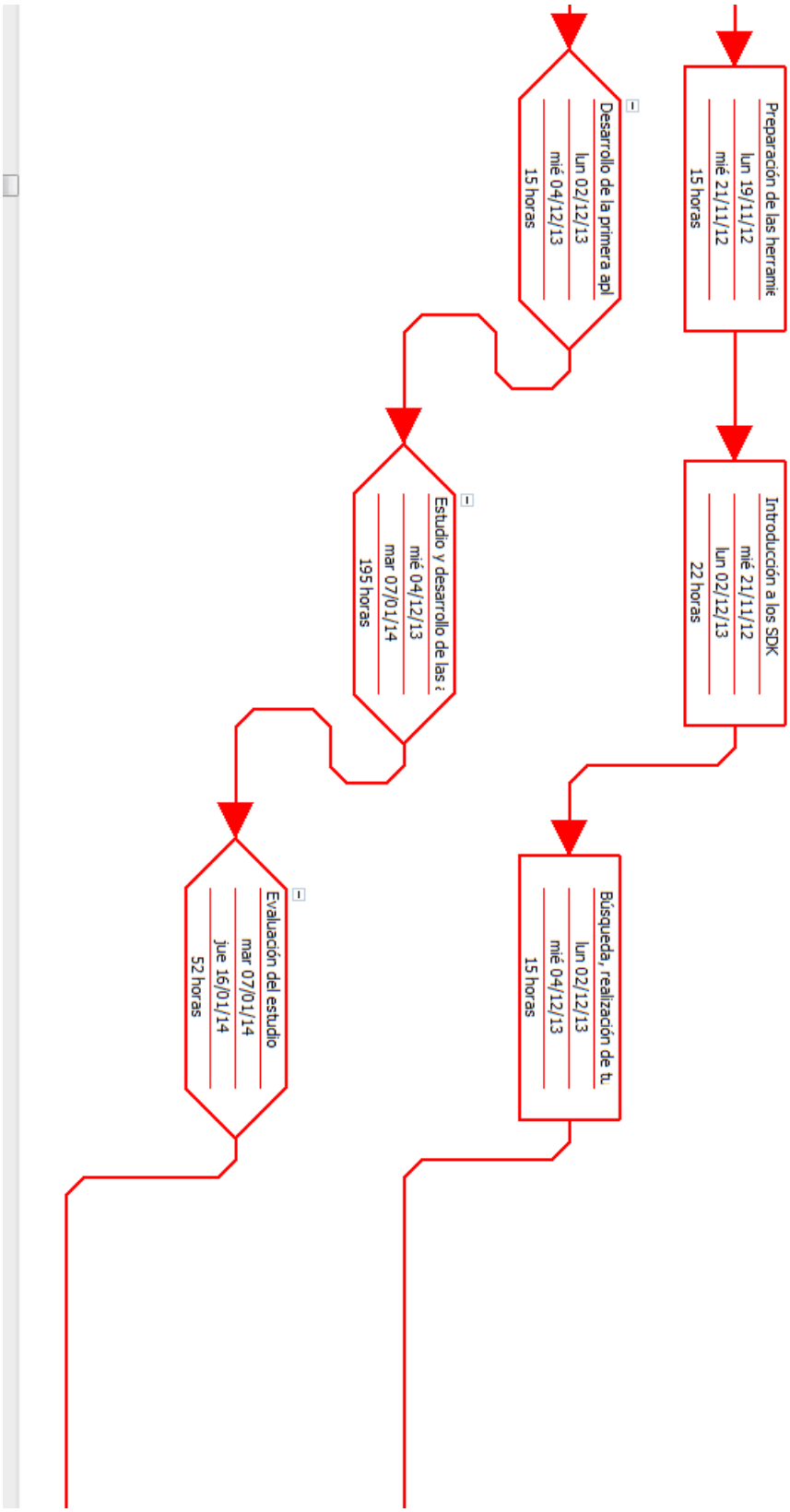


Figura A.2: Diagrama de Pert del Proyecto

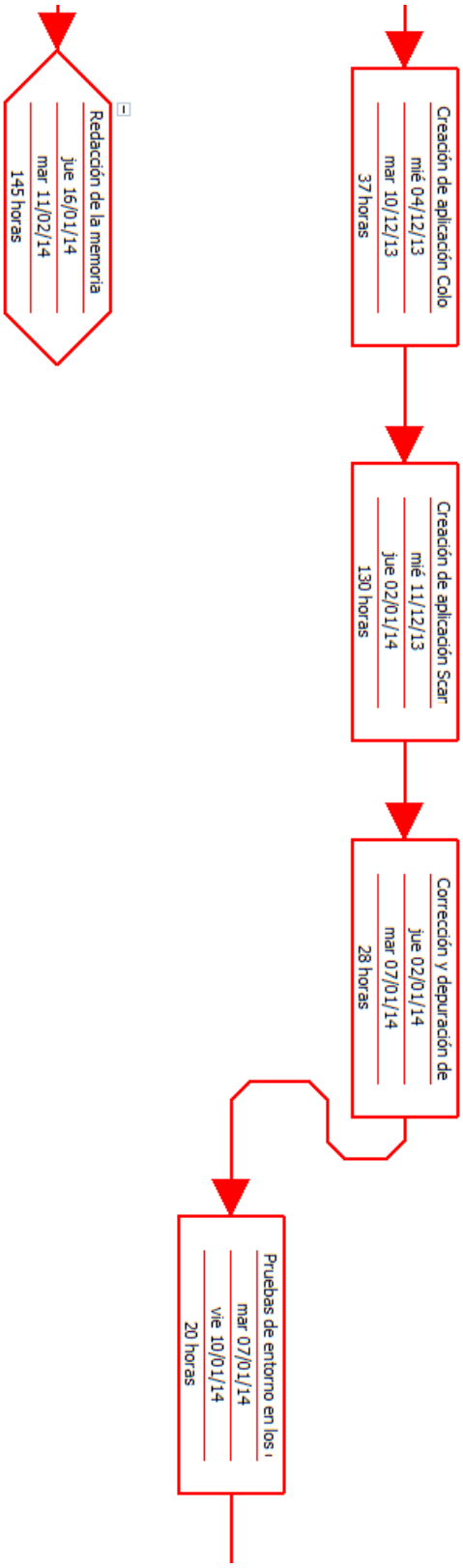


Figura A.3: Diagrama de Pert del Proyecto

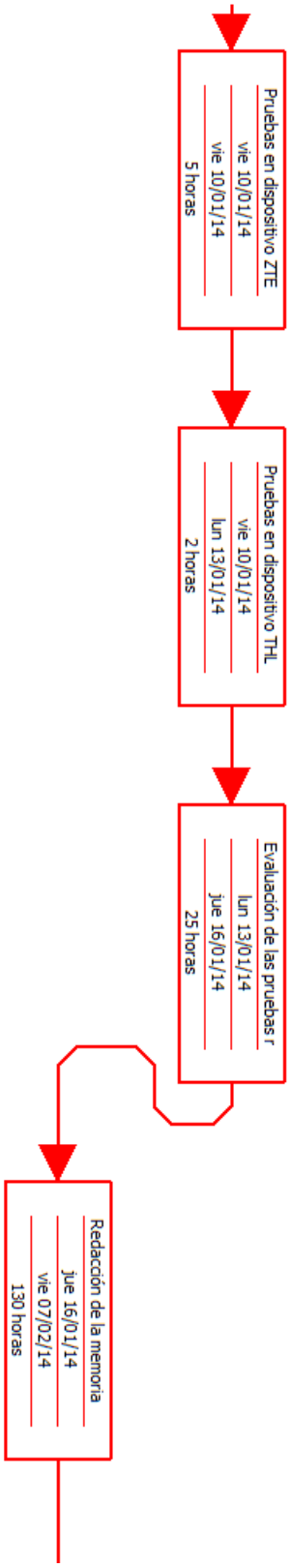


Figura A.4: Diagrama de Pert del Proyecto

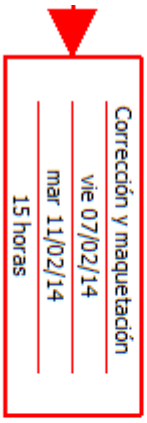


Figura A.5: Diagrama de Pert del Proyecto

ANEXO B

HolaMundo en WebOS

B.1. HolaMundo en WebOS -continuación-

En esta sección se comentarán, más detenidamente, los pasos que hay que seguir para comenzar a crear la aplicación "HolaMundo".

Como se indicó en su correspondiente capítulo (3.1), hay que tener en cuenta que cuando se crea una aplicación en el sistema operativo *WebOS* se puede realizar de dos maneras distintas:

Si se emplea el sotfware *Eclipse* con el *plug-in WebOS SDK*, se ha de seleccionar: *File ->New ->Basic Application ->New Mojo Application* para comenzar a crear una nueva aplicación.

Después se introducirá el nombre del proyecto y el título, en este caso, *HolaMundo*, y se aceptarán los valores predeterminados para los demás campos. Haciendo *click* en *Finalizar* se creará la aplicación marco. De esta manera, *Eclipse* creará automáticamente los ficheros necesarios para una aplicación de este tipo, estableciendo la estructura de los directorios anteriormente.

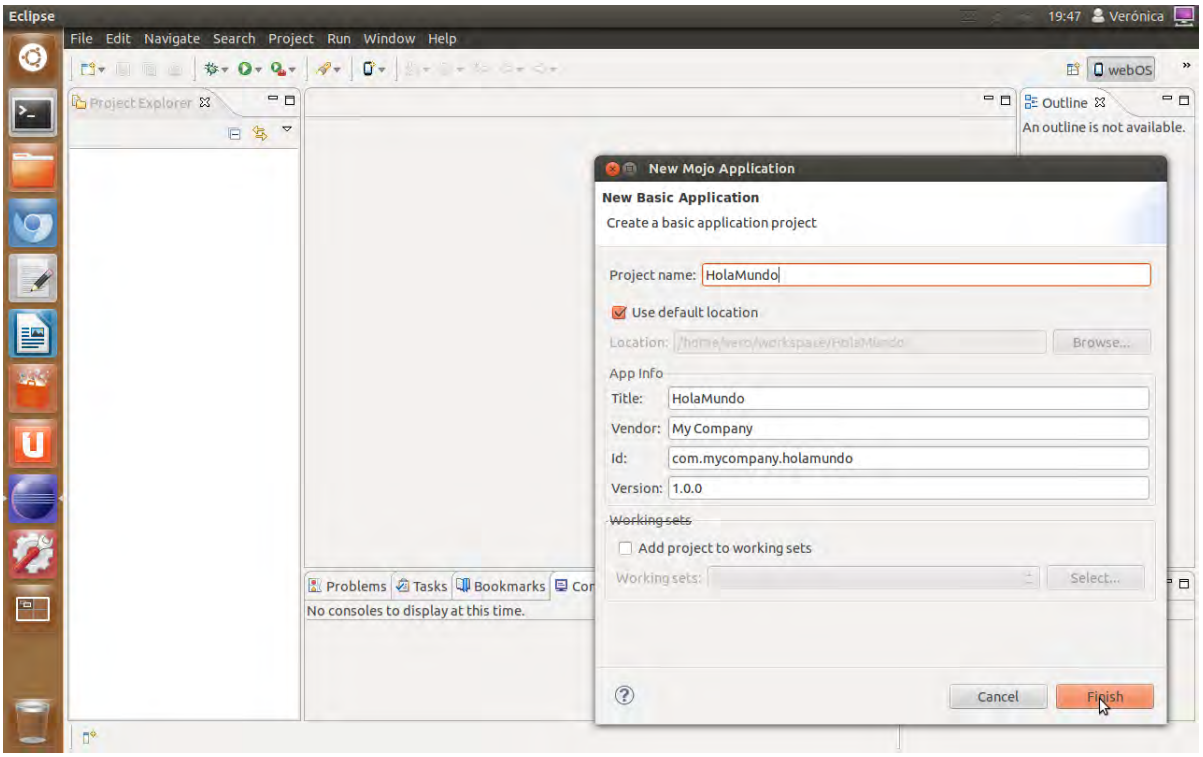


Figura B.1: Creación de la aplicación HolaMundo mediante Eclipse

Como se comentó anteriormente, si en vez de emplear este software, se desea hacer de manera manual y únicamente utilizar un editor de textos y la ventana de comandos, es totalmente viable, aunque por una mayor comodidad para el usuario se recomienda el uso de *Eclipse*. De todos modos, a continuación también se describirá cómo crear manualmente esta aplicación sin tener que emplear este software:

En este caso, el primer paso sería abrir la ventana de símbolo del sistema (ventana de comandos) y crear un directorio que será el espacio de trabajo (por ejemplo, *APP_DIR* o *workspace* creado por defecto en *Eclipse*) para el desarrollo de aplicaciones. Desde ese mismo directorio, habría que escribir (nótese las comillas simples dentro de las comillas dobles):

palm-generate -p "{title:'Hola Mundo', id:com.mystuff.holamundo, version:'1.0.0'}" HolaMundo

Creación de la aplicación

Método 1: Iniciar el emulador

En primer lugar, dependiendo del sistema operativo que se emplee se deberá de seguir unas pautas u otras a la hora de iniciar el emulador:

- *Linux:* En el símbolo del sistema, habría que escribir *palm-emulator*.

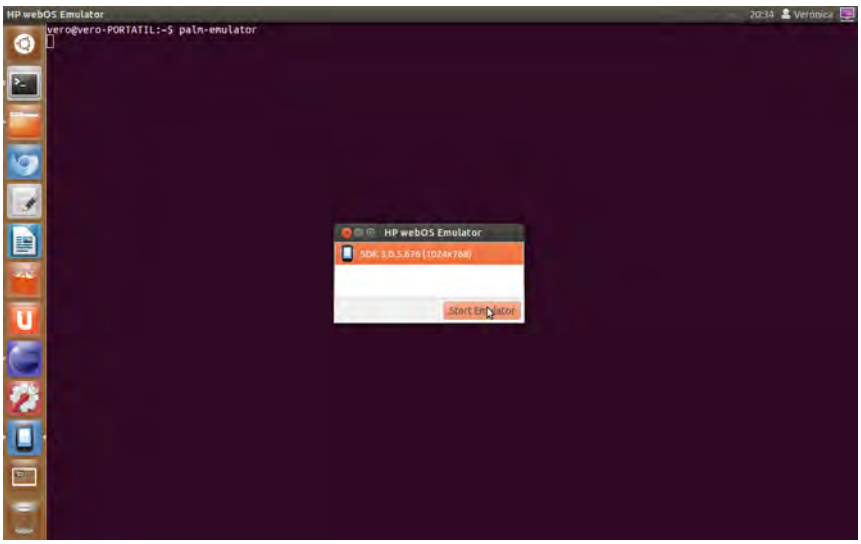


Figura B.2: Inicio del emulador desde línea de comandos

- *Mac OS:* En la carpeta *Applications*, habría que hacer *doble click* en el icono del emulador.
- *Windows:* Seleccionar *Start (Inicio) –>All Programs (Todos los Programas) –>Palm –>SDK emulador*.

A continuación, se lleva a cabo la ejecución de las aplicaciones en el emulador.

Método 2: Iniciar el emulador mediante Eclipse

Si se utiliza *Eclipse* con el *plug-in* adecuado de HP *WebOS*, se puede ejecutar una aplicación en el emulador. Para ello se debe seleccionar:

Run –>Run Configurations...

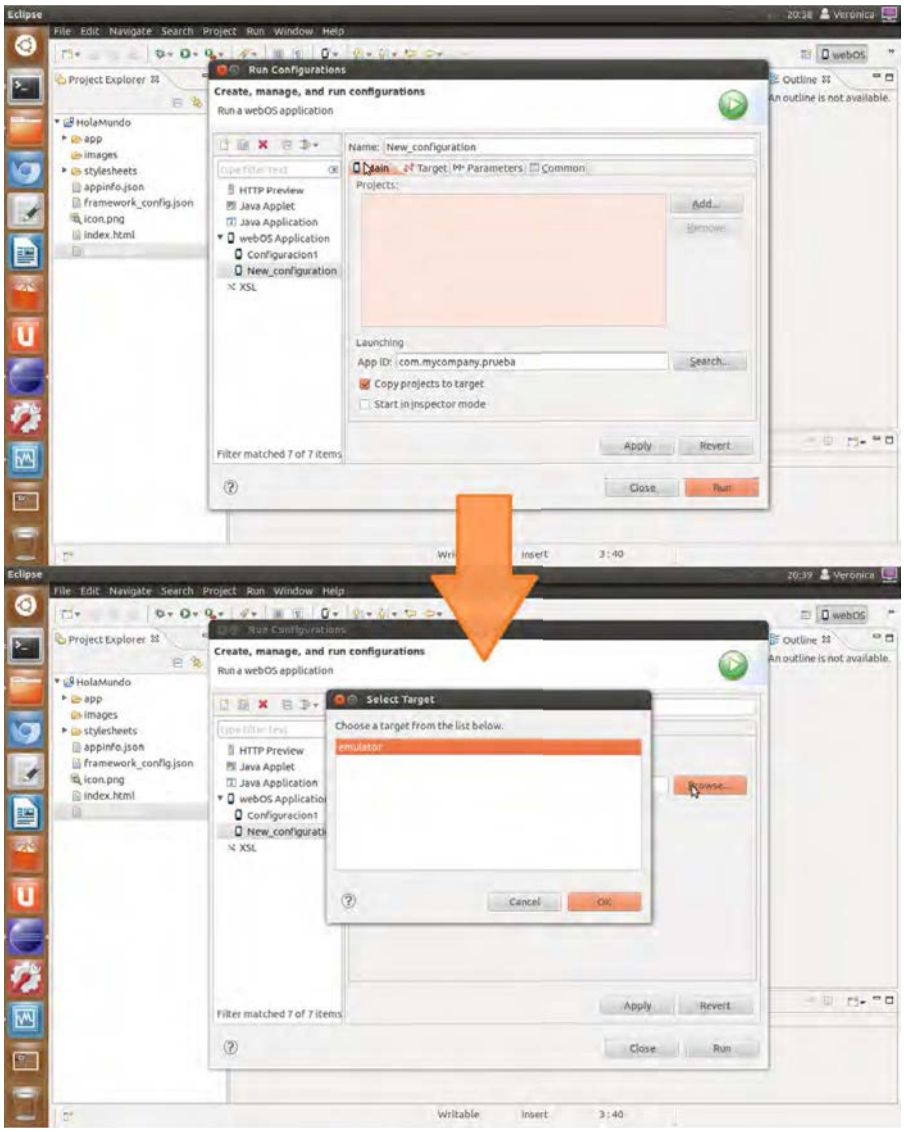


Figura B.3: Creando una nueva configuración

Después habría que seleccionar *Emulador de Palm*. En ejecuciones posteriores, basta con seleccionar *Run –>Run As... –>Mojo Application*. A partir de ahí, *Eclipse* automáticamente instalará paquetes y ejecutará la aplicación.

Con este software, no hay necesidad de desinstalar la aplicación antes de que ésta vuelva a correr, por lo que a continuación, se detallará qué acciones hay que seguir, empleando el Método 1 -línea de comandos-.

Método 1: Iniciar, Empaquetar e Instalar la aplicación en el emulador mediante línea de comandos

Si, por el contrario, se está empleando las herramientas de línea de comandos, se deberá ejecutar la aplicación en el símbolo del sistema utilizando las herramientas que se detallarán a continuación:

En primer lugar, se ha de abrir una ventana del símbolo del sistema y, a continuación, ir al directorio de espacio de trabajo. Después, hay que emplear el comando de *palm-package* de la siguiente manera para empaquetar el directorio que contiene la aplicación:

palm-package HolaMundo

En segundo lugar, se ha de emplear el comando *palm-install* del siguiente modo para instalar el archivo resultante .ipk en el emulador.

NOTA: *Se ha de tener en cuenta que no está el emulador en funcionamiento, pero hay un dispositivo WebOS conectado en el puerto USB del ordenador, se instalará el paquete del dispositivo si se ha habilitado el "modo de programador". Podrá encontrarse el icono de la aplicación en la parte inferior del menú de inicio. Además, al conectar el dispositivo al ordenador para cargar una aplicación, se deberá seleccionar la opción "Sólo carga" en el dispositivo.*

palm-install com.mystuff.holamundo_1.0.0_all.ipk

Método 1: Iniciar la aplicación en el símbolo del sistema

Para iniciar la aplicación desde línea de comandos, se ha de iniciar el emulador, si no se ha hecho anteriormente. Después, se deberá abrir una ventana del símbolo del sistema y, a continuación, habrá que indicar el directorio de espacio de trabajo. Con el siguiente comando se iniciará la aplicación automáticamente:

palm-launch com.mystuff.holamundo

Método 1: Cerrar la aplicación en el símbolo del sistema

Con el siguiente comando se cerrará la aplicación automáticamente:

palm-launch -c com.mystuff.holamundo

Método 1: Eliminar la aplicación en el símbolo del sistema

Para la eliminación de la aplicación desde línea de comandos, se ha de introducir el siguiente comando:

palm-install -r com.mystuff.holamundo

Después de empaquetar e instalar la aplicación en el emulador y una vez se han escrito las líneas correspondientes en el fichero *.html* para que se pueda visualizar el texto, se lanzará la aplicación en la línea de comandos o mediante el emulador seleccionando: *Hola Mundo* (el icono por defecto es una luna creciente). La visualización en el emulador será la siguiente:

Figura B.4: Aplicación HolaMundo ejecutándose en Simulador WebOS

A continuación, en esta sección se recordará la estructura de la aplicación, mientras que en la sección posterior se indicarán los problemas más destacados que han surgido durante su desarrollo.

Estructura de la aplicación

Una vez llegado aquí, se podrá observar el contenido de la nueva carpeta HolaMundo, generada en *APP_DIR*. Indicar que partir de línea de comandos se generan archivos más básicos que los que se generan en *Eclipse* automáticamente en su directorio de trabajo *workspace*. A continuación se muestran los archivos y directorios que se crean en cada método a modo recordatorio:

Método 1: Línea de comandos -APP_DIR-

- *Carpeta de imágenes* - Contiene las imágenes que esta aplicación emplea.
- *Carpeta de hojas de estilo (CSS) y JavaScript* - Contiene archivos referentes a hojas de estilo en cascada que utiliza la aplicación, así como archivos de tipo *JavaScript*.
- *appinfo.json* - Archivo que contiene información de la aplicación.
- *depends.js* - Archivo que direcciona los ficheros de tipo *CSS* y *JavaScript*.
- *framework_config.json* - Archivo que contiene la configuración del marco.
- *index.html* - Escenario principal en el que aparecerán las escenas de la aplicación.

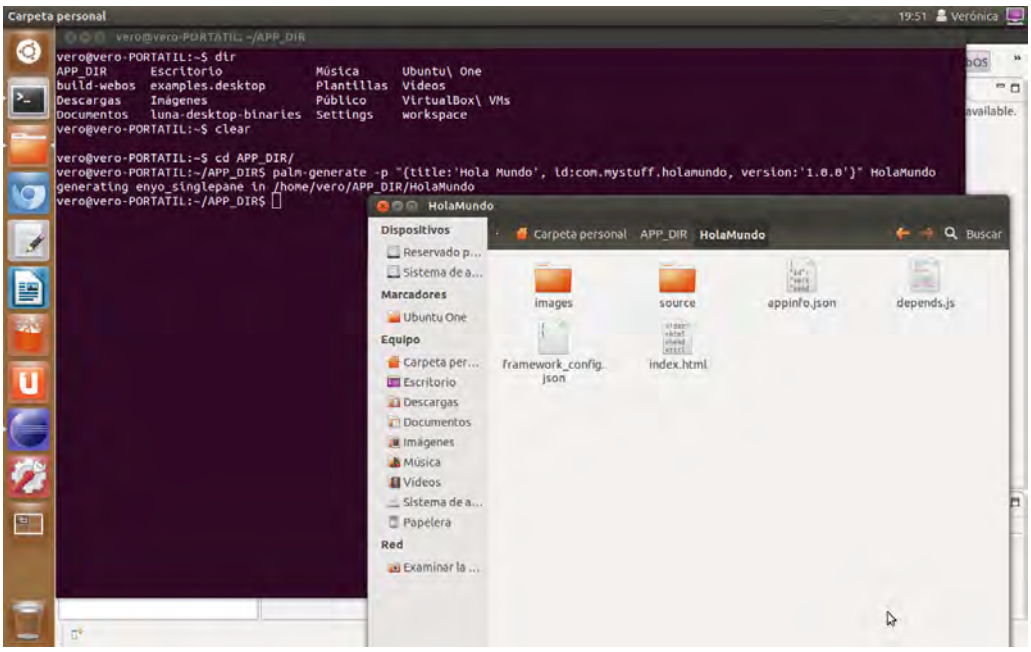


Figura B.5: Contenido del espacio de trabajo (APP_DIR)

Método 2: Eclipse -workspace-

- *Carpeta de la aplicación -app-* - Contiene los asistentes, modelos y vistas que componen la aplicación. De manera automática se agregan los archivos correspondientes a este directorio.
- *Carpeta de imágenes* - Contiene las imágenes que esta aplicación emplea.
- *Carpeta de hojas de estilo (CSS)* - Contiene archivos referentes a hojas de estilo en cascada que utiliza la aplicación.
- *appinfo.json* - Archivo que proporciona información que el marco SDK utiliza para empaquetar y ejecutar la aplicación.
- *framework_config.json* - Archivo que contiene la configuración del marco.
- *icon.png* - Imagen de inicio que se muestra en el emulador o dispositivo.
- *index.html* - Escenario principal en el que aparecerán las escenas de la aplicación.
- *sources.json* - Lista de los archivos de origen para cada escena.

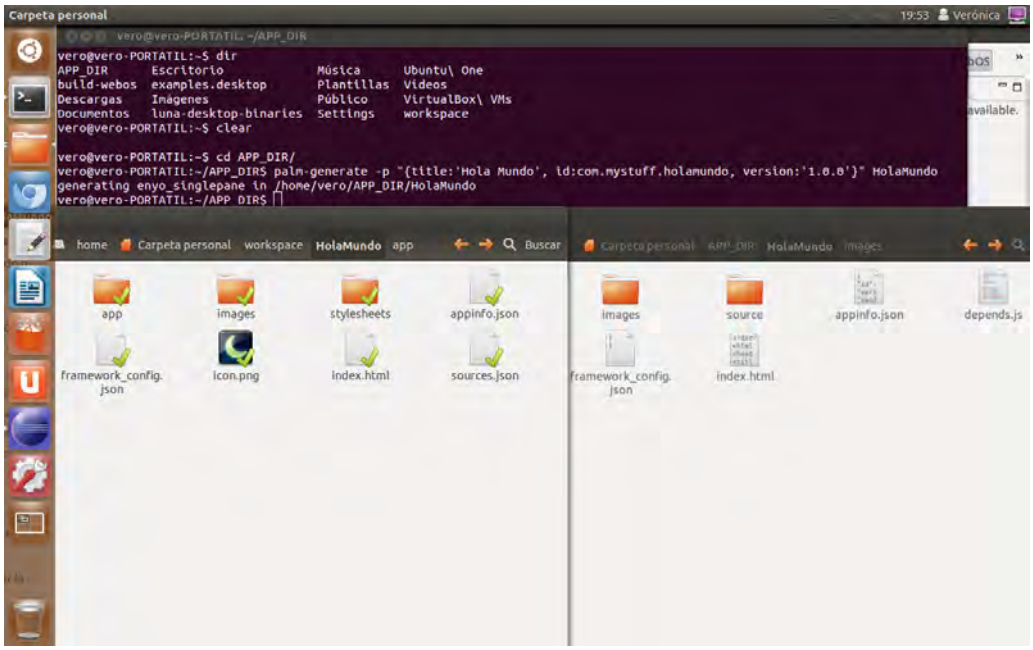


Figura B.6: De izquierda a derecha se muestra el contenido de los directorios workspace y APP_DIR

Escenario

Se define como escenario a la plataforma sobre la que se construye la interfaz de usuario de la aplicación. Un escenario generalmente se corresponde a una única ventana de la aplicación. Las aplicaciones más simples tienen un solo escenario, contenido en el archivo *index.html*. En cambio, una aplicación que permite al usuario realizar varias acciones al mismo tiempo puede requerir más de uno. Por ejemplo, una aplicación de correo electrónico podría mostrar la bandeja de entrada en un mismo escenario, pero será necesaria una segunda fase para redactar un mensaje de correo electrónico nuevo.

Cabe destacar que este archivo (*index.html*) conforma la página *HTML* estándar a partir de la etiqueta <script>necesaria para las aplicaciones *WebOS*.

Escena

Una escena es una pantalla con formato en la que se muestra la presentación de información de la aplicación o una tarea para el usuario. Cada escena tiene una vista y un asistente. La vista determina el diseño y apariencia de la escena, mientras que el asistente determina el comportamiento. Algunas escenas también tienen modelos, que suministran datos.

Cabe indicar que el comando *palm-generate* se puede utilizar para crear escenas y asistentes y el archivo *sources.json* correlaciona las escenas y asistentes.

De todos modos, existen algunos inconvenientes que han de destacarse si se decide emplear el primero de los métodos:

1. Los cambios realizados durante el desarrollo y la depuración no aparecerán, a menos que se elimine primero la versión anterior y, a continuación, se vuelva a empaquetar e instalar la aplicación actualizada.

2. La instalación de una nueva versión de una aplicación, no elimina los archivos de origen que no están presentes en la nueva versión. Por lo tanto, durante las pruebas, es útil eliminar la versión anterior de una aplicación antes de instalar una nueva.

3. Y, por último, no hay que olvidar empaquetar e instalar la aplicación en el emulador para que la aplicación funcione de la manera deseada.

B.2. Errores a tener en cuenta

A continuación se detallan los errores más comunes que se han cometido durante esta primera toma de contacto con *WebOS*, con el fin de que al usuario le sea más fácil y rápido avanzar si le sucede alguno de estos problemas:

Eclipse no puede arrancar

Para solventar este error[1], hay que observar que la ruta o también denominada *PATH* donde se encuentra el ejecutable de *Eclipse* es la correcta. Para ello, dentro de la carpeta personal (*home*) se encuentra el fichero *.bashrc*. Se procede a abrir éste, y al final del todo hay que incluir la siguiente línea, de no ser así, eclipse no podrá arrancar.

export PATH=~/Descargas/eclipse/:$PATH para Sistemas Operativos Linux.

export PATH=~/Descargas/eclipse/;$PATH para Sistemas Operativos Windows.

Cabe destacar, que dependiendo de si es un sistema operativo u otro, la sentencia va con dos puntos o con punto y coma.

Eclipse no encuentra el emulador

A la hora de configurar el arranque del emulador dentro de *Eclipse*, es decir, dentro de *Run Configurations*, cuando se escoge el proyecto, en la pestaña *Target*, debe dejar de elegir en el apartado de *Device name* el emulador donde se desee que corra la aplicación.

Si se presenta el caso de que no encuentra el emulador, es decir, que no aparezca a la hora de seleccionarlo, aún pulsando en el botón *Browse* y que sólo de la opción de escoger *Use default device* y se deje esta opción indicada, cuando se desee arrancar el programa (pulsando *Run*), saltará el error: *'Launching New_ configuration' has encountered a problem. Unable to find default device.*

[1] ***Error:*** *A Java Runtime Environment (JRE) or Java Development Kit (JDK) must be available in order to run Eclipse. No Java virtual machine was found after searching the following location*

El problema que tiene es que no encuentra el emulador ya que éste no está activo, por lo que la solución es tan obvia como tener el emulador abierto antes para que eclipse pueda encontrarlo. De esta manera, cuando se pulse sobre el botón *Browse* aparecerá el emulador como en la siguiente figura:

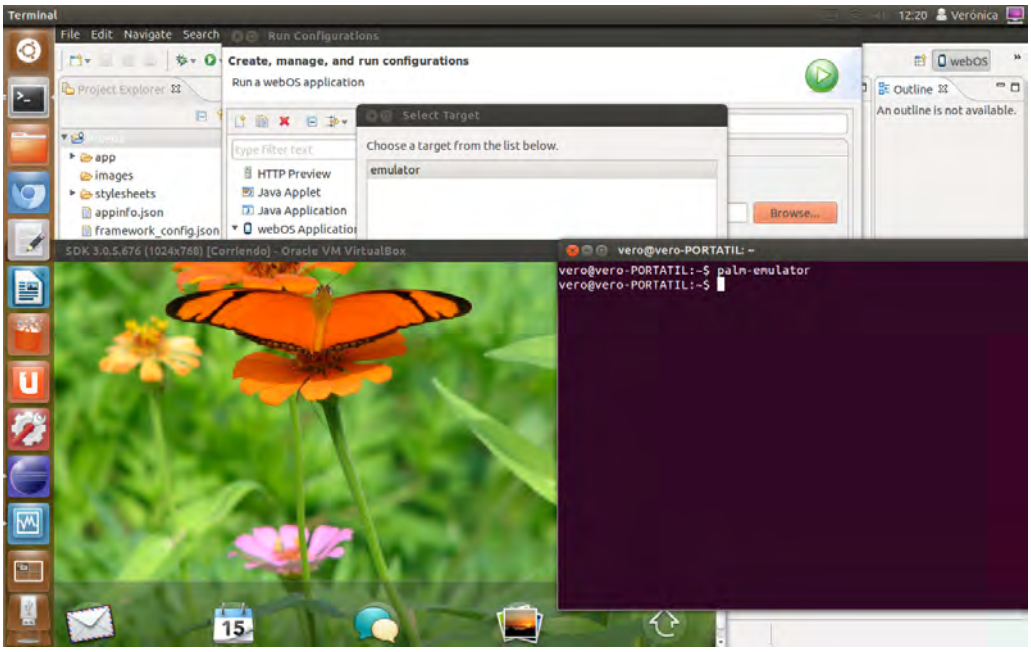


Figura B.7: El emulador aparece como opción

ANEXO C

HolaMundo en Firefox OS

C.1. HolaMundo en Firefox OS -continuación-

En el caso de *Firefox OS*, para crear la aplicación "HolaMundo" habrá que crear los siguientes ficheros:

- Un archivo .html donde se pone el nombre de la aplicación, en este caso "HolaMundo", usando la semántica sencilla de *HTML 5*.
- Una imagen con dimensiones de 128x128 que sirve como icono de la aplicación.
- Un archivo de manifesto con extensión .webapp (*ejemplo: manifest.webapp*). Este documento tiene una estructura JSON donde se aportan algunas características de la aplicación como el nombre, la descripción de la aplicación, los iconos que emplea,...

Una vez creada ésta, para poder ejecutar la aplicación en el simulador, hay que abrir el navegador Firefox e ir al menú *Herramientas ->Desarrollador Web ->Firefox OS Simulator*. Después, se localiza el botón con la opción *Add Directory*, se procede a buscar la carpeta donde se a almacenado el proyecto y, a continuación, se selecciona el archivo manifest para abrirlo.

Cabe destacar, que las emulaciones se realizan sobre ambos simuladores de Firefox por las razones que se indican en el capítulo 3.1.

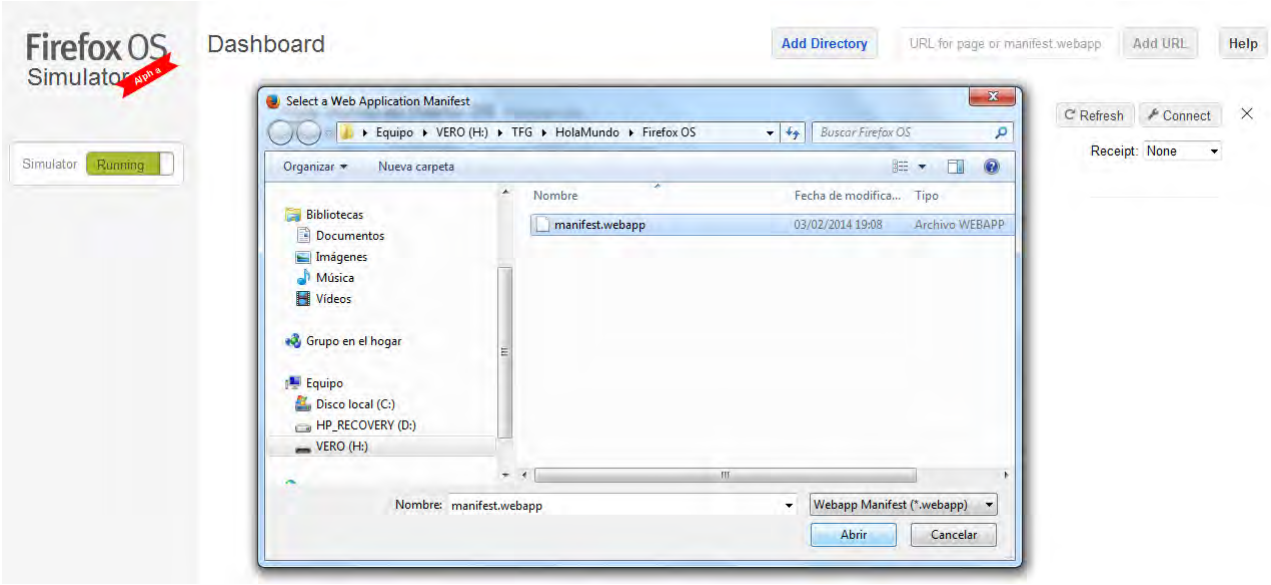


Figura C.1: Insertar archivo manifest en el simulador de Firefox OS

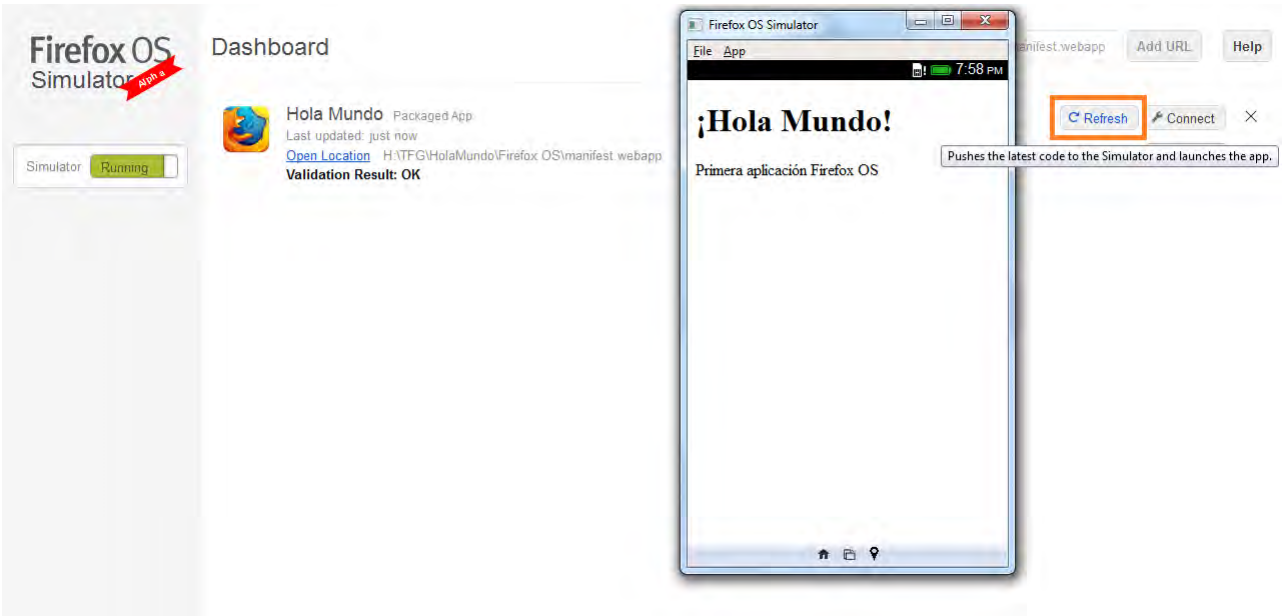


Figura C.2: Emulando aplicación en el simulador de Firefox OS

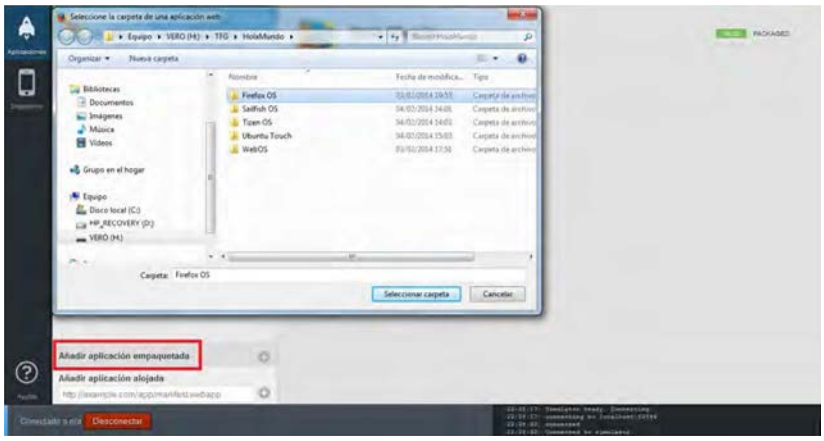


Figura C.3: Añadiendo el proyecto de la aplicación HolaMundo en App Manager

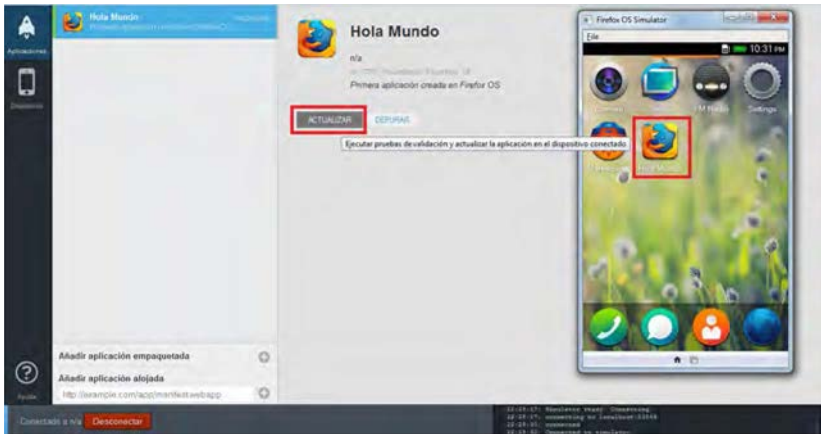


Figura C.4: Mostrando icono de la aplicación en el simulador App Manager

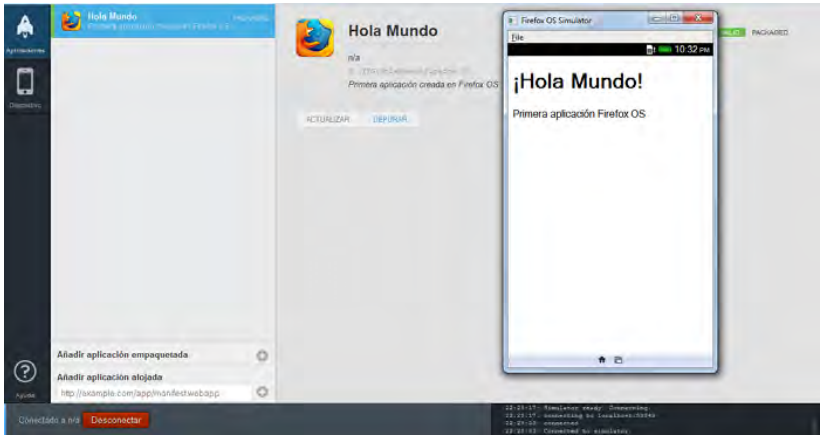


Figura C.5: Emulando aplicación en el simulador App Manager

ANEXO D

HolaMundo en Tizen OS

D.1. HolaMundo en Tizen OS -continuación-

En esta sección, como se comentó anteriormente, se indicarán, más detenidamente, los pasos que hay que seguir para comenzar a crear la aplicación "HolaMundo" en Tizen OS. Para ello, en primer lugar, mediante la instalación de imágenes SDK hay que descargar el *administrador de instalación SDK* y después ejecutarlo. Indicar que hay que tener en cuenta el sistema operativo en el que se va a ejecutar el emulador:

- *Ubuntu:* Antes de ejecutar el administrador de instalación del SDK, hay que agregar un permiso ejecutable al administrador de instalación con el siguiente comando: *$ chmod +x tizen-sdk–ubuntu.bin*
- *Mac OS:* Abrir *tizen-sdk–macos64.dmg* y hacer *click* en instalar SDK de Tizen.
- *Windows:* Instalar el SDK de Tizen descargado.

Una vez se a iniciado el Asistente de instalación del SDK:

1. Hacer *click* en Avanzado.
2. En la ventana de configuración avanzada, seleccionar el botón imagen SDK.
3. Hacer *click* en el botón de la carpeta, buscar el archivo de imagen SDK y hacer *click* en Aceptar.

4. Hacer *click* en Siguiente.

5. Aceptar los términos y condiciones y hacer *click* en Siguiente.

6. Seleccionar los componentes que se desean instalar y hacer *click* en Siguiente.

7. Seleccionar la carpeta de inicio del SDK y hacer *click* en Instalar.

Indicar que el Asistente de configuración SDK informa cuando la instalación ha sido completa. Eso significa que el SDK está totalmente instalado y configurado en el equipo.

Tras instalar el SDK de *Tizen OS*, se crea automáticamente el *Tizen IDE* en *Eclipse*. Dicho entorno de desarrollo es parte del SDK y va integrado con él. Cuando se inicia el IDE, pide que se defina el trayecto para el espacio de trabajo.

Al igual que ocurre en la instación del SDK, el inicio de este entorno de desarrollo integrado variará dependiendo del sistema operativo donde se desee emular la aplicación:

- *Ubuntu:* Aplicaciones –>SDK de Tizen –>IDE de Tizen.
- *Mac OS:* Ir al directorio *tizen-sdk/ide* y ejecutar la aplicación *Tizen IDE*.
- *Windows:* Inicio –>SDK de Tizen –>IDE de Tizen.

NOTA: *Indicar que al iniciar el IDE por primera vez, aparece una página de bienvenida con más información sobre Tizen y el desarrollo con el IDE. En esta página, habrá que hacer click en el icono de la mesa de trabajo para ir a la vista del desarrollo.*

Para comenzar a crear esta aplicación, se ejecuta dicho entorno de desarrollo (*Tizen IDE*) y se siguen las pautas que se indican a continuación:

En primer lugar, se selecciona *File –>New –>Tizen Web Project*:

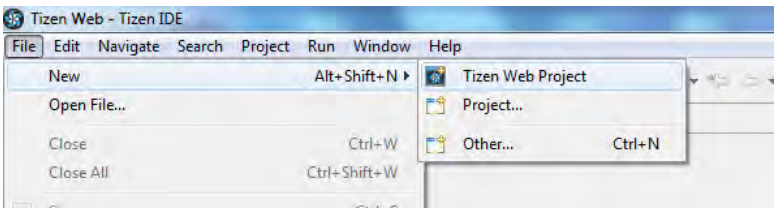


Figura D.1: Creando un nuevo proyecto en Tizen OS

Después, se crea la aplicación según la siguiente figura:

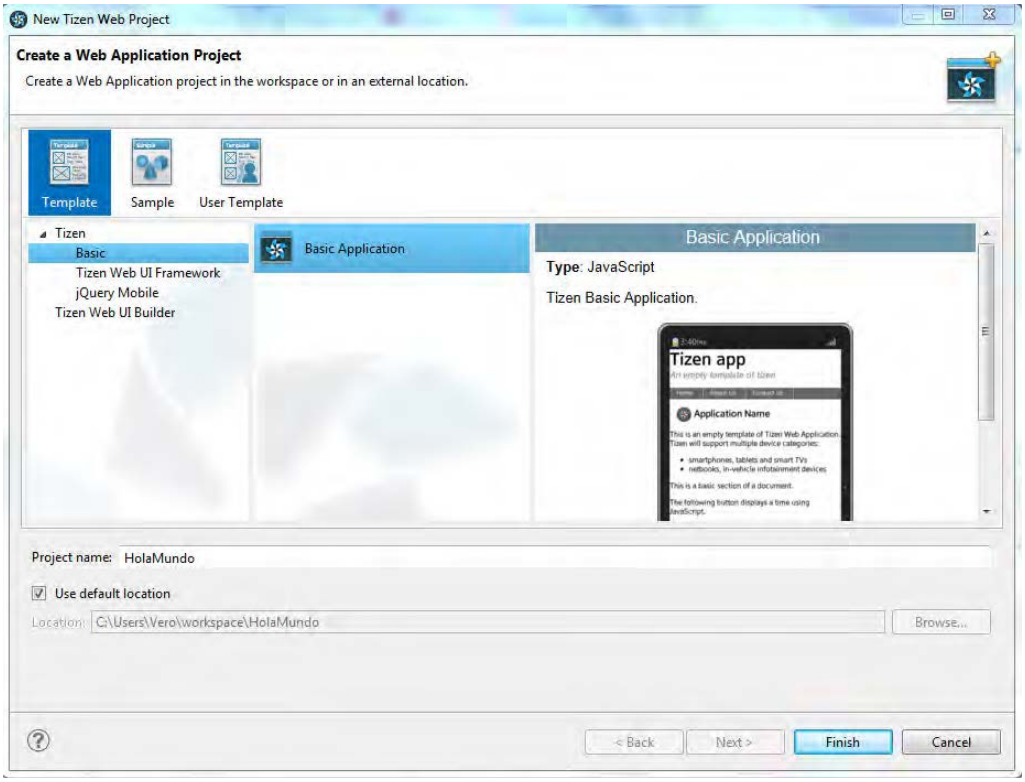


Figura D.2: Creando primera aplicación Tizen OS

Una vez creado el proyecto, se construye éste para emularlo:

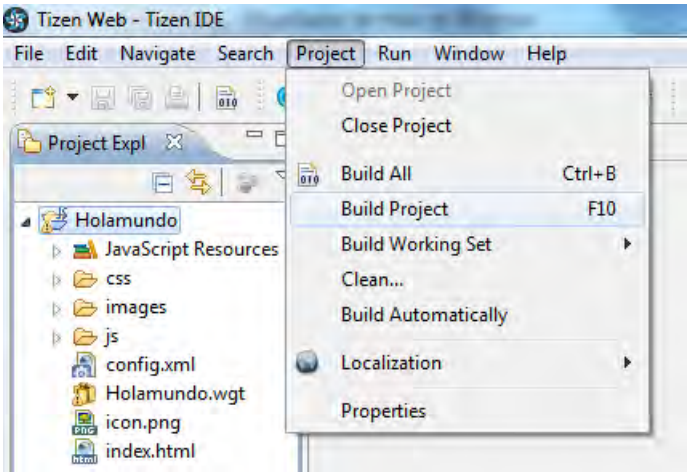


Figura D.3: Construyendo el proyecto para emularlo

Por último, se ejecuta el simulador para verificar que se ha creado correctamente la aplicación:

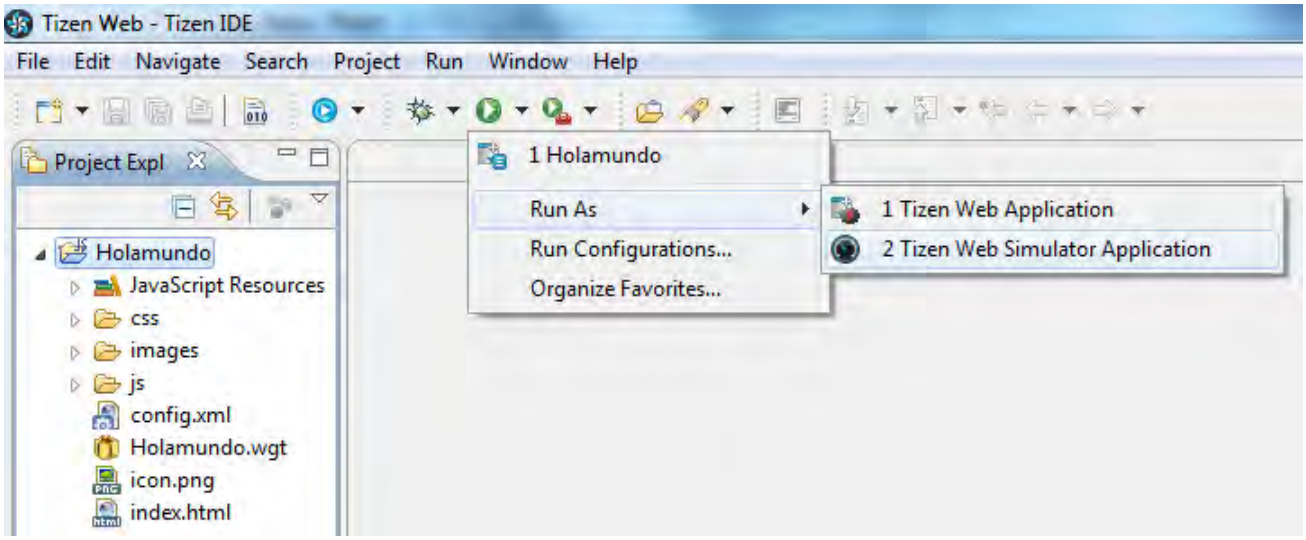


Figura D.4: Ejecutando el Simulador de Tizen OS

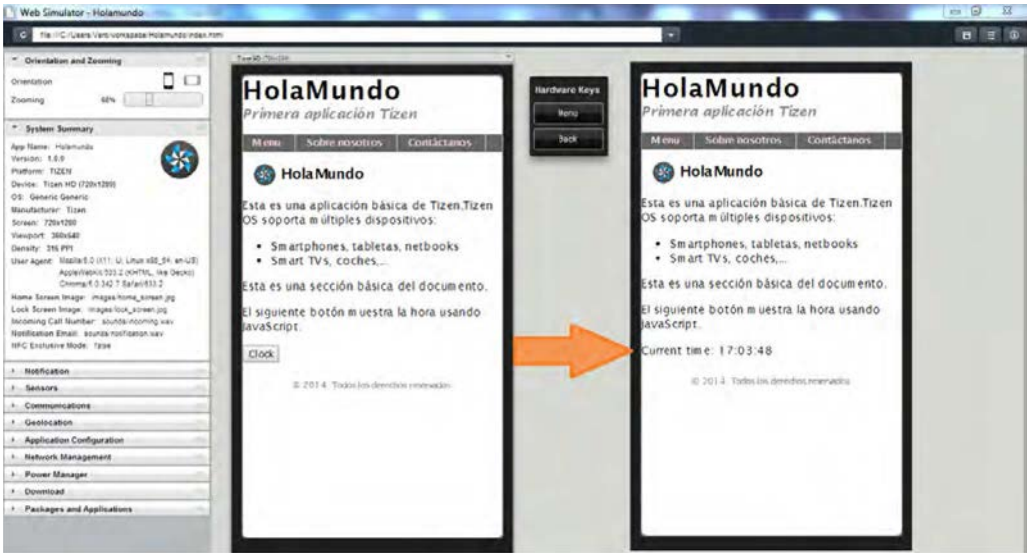


Figura D.5: Ejecutando la aplicación Holamundo en el Simulador de Tizen OS

ANEXO E

HolaMundo en Sailfish OS

E.1. HolaMundo en Sailfish OS -continuación-

Al igual que en las anteriores secciones, en ésta se procede a crear la aplicación "HolaMundo" pero en este caso, en el sistema operativo Sailfish. En primer lugar, lo primero que hay que hacer es instalar su SDK. En el caso de *Sailfish OS*, para la instalación de éste, ocurre lo mismo que en *Tizen OS*, el proceso es distinto dependiendo del sistema operativo donde se desee instalar éste [40]:

- *Linux:*

  1. Descargar el archivo de ejecución (32 bits o 64 bits, dependiendo del sistema host). **Sugerencia:** Para conocer el tipo de host escribir en un terminal: *getconf LONG_BIT*.
  2. Abrir el Terminal para proporcionarle permisos de ejecución a la instalación:

     ```
     $ chmod +x ~/Downloads/<installer_name>
     ```

  3. Ejecutar el programa de instalación como un usuario normal (**no como usuario root**):

     ```
     $ ./Downloads/<installer_name>
     ```

     Por defecto se instalará *Sailfish OS* en el directorio personal.

- *Mac OS:*

  1. Descargar paquete *SDK Installer*.
  2. Abrir el paquete haciendo doble *click* en el icono.
  3. Abrir la aplicación *Installer SDK* encontrado en el paquete.

- *Windows:*

  1. Descargar la aplicación *SDK Installer* a su carpeta de descargas y localizarla. Cabe destacar que las plataformas que están soportadas actualmente son: XP/Vista y Windows 7.
  2. Abrir el ejecutable haciendo doble *click* en la aplicación *Installer SDK*.

Indicar que a partir de aquí, el flujo de la instalación ya es común. Cuando la ventana del instalador SDK se abra, hacer *click* en Siguiente, leer y aceptar el contrato de licencia. Después, seguir el asistente de configuración con los valores por defecto y, por último, finalizar éste. Después se eso, *Sailfish OS IDE* se iniciará automáticamente una vez que salga del asistente de configuración.

Por lo que ya se puede crear un nuevo proyecto, para ello ir a (*File –>New File or Project...*), siguiendo los pasos que se muestran a continuación:

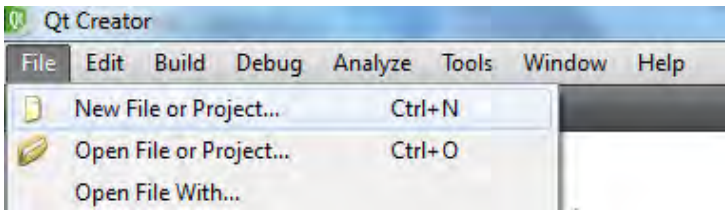


Figura E.1: Creando un nuevo proyecto en Sailfish OS

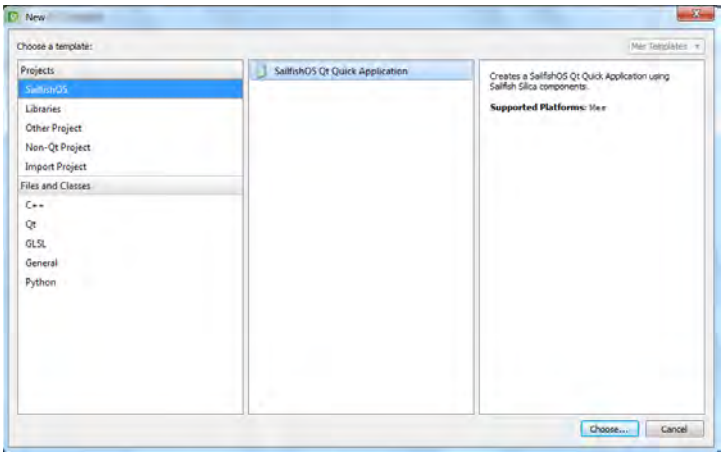


Figura E.2: Creando la primera aplicación en Sailfish OS

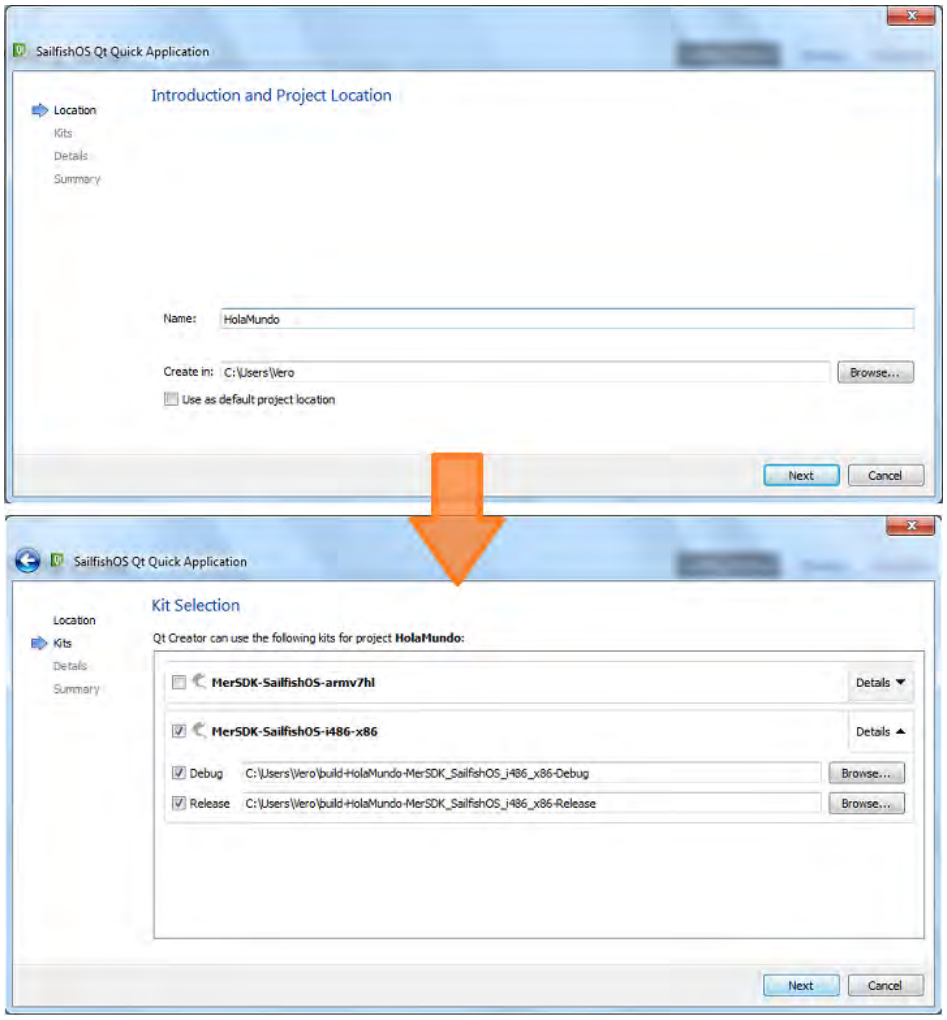


Figura E.3: Dando nombre a la aplicación y seleccionando el tipo de sistema

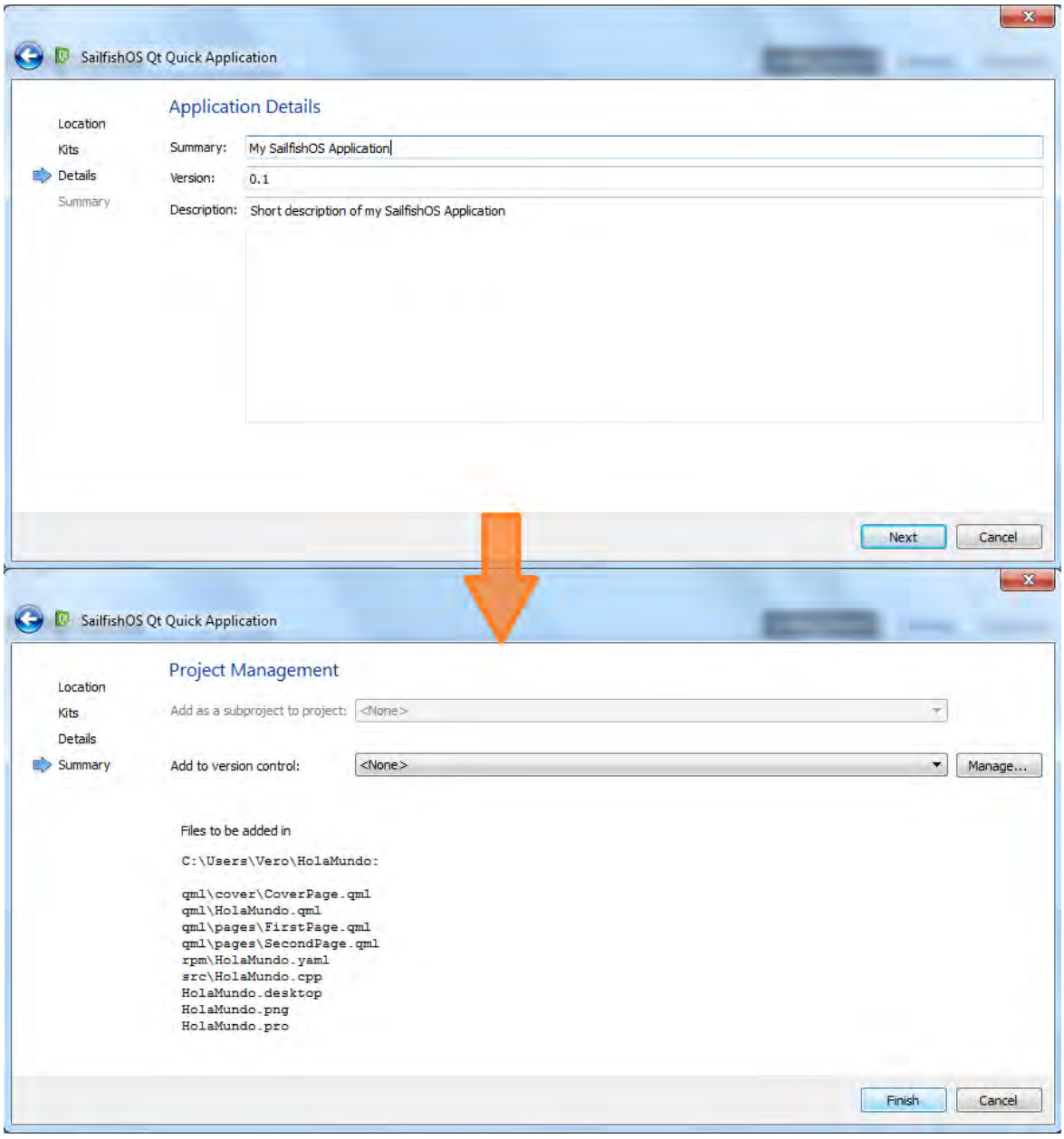


Figura E.4: Finalizando la creación del proyecto HolaMundo en Sailfish OS

NOTA: *El SDK de Sailfish OS emplea una máquina de compilación denominada "Mer" y una máquina virtual para ejecutar el emulador. Si estas dos máquinas no están funcionando cuando se intenta construir o desplegar una aplicación, se indicará que se inicien ambas antes.*

Cabe destacar que cuando un proyecto de este entorno se encuentra abierto, el SDK muestra automáticamente dos botones de control en la barra lateral izquierda para iniciar / detener la

máquina "Mer" y el emulador. Por lo que para ejecutar la aplicación en el emulador, se debe de activar el SDK de *Sailfish OS*, *Start Sdk* y, después, activar el emulador (*Start Emulator*), de tal manera que estén las dos ejecutadas a la vez para que la aplicación se ejecute correctamente en el simulador como se comentó en la **NOTA** anterior:

Figura E.5: Proceso para activar SDK y emulador - *Antes de la conexión* (símbolos en verde), *Conectando* (símbolos en gris) y *Conexión establecida* (símbolos en rojo).

Una vez se ha establecido la conexión con el simulador, se procede a emular la aplicación creada:

Figura E.6: Simulando aplicación Holamundo en Sailfish OS

ANEXO F

HolaMundo en Ubuntu Touch OS

F.1. HolaMundo en Ubuntu Touch OS -continuación-

En esta sección se procede a crear esta aplicación básica en *Ubuntu Touch OS* como primera toma de contacto con este entorno.

Lo primero que hay que hacer es instalar el SDK [41], para ello se ha de poner la siguiente línea de comando en el terminal: *sudo apt-get update && sudo apt-get install ubuntu-sdk*. Cabe indicar que el emulador de este sistema operativo móvil requiere ser ejecutado en Linux, en este caso se emplea más concretamente Ubuntu como sistema operativo.

Por otro lado, el software que se instala junto con el SDK para crear aplicaciones es el *QT Creator*. A continuación se indican las pautas que se siguieron para crear la aplicación "HolaMundo".

En primer lugar, se ejecuta *QT Creator*, después en *File ->New* se selecciona el tipo de proyecto, en este caso *Ubuntu* y *HTML5 Touch UI* como se muestra en la figura:

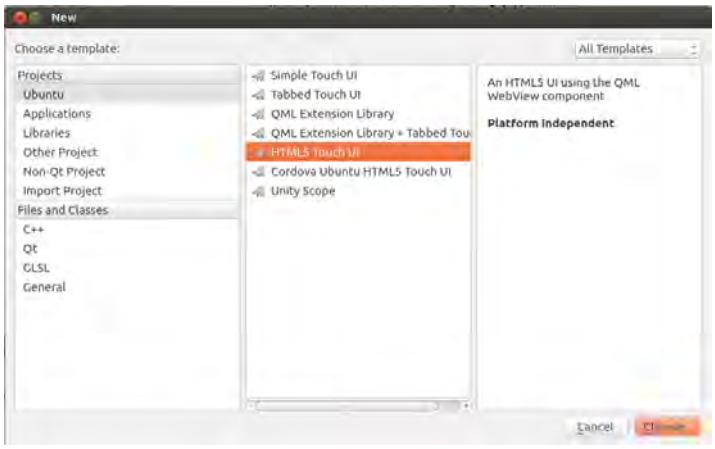


Figura F.1: Seleccionando el tipo de proyecto

Se da nombre al proyecto:

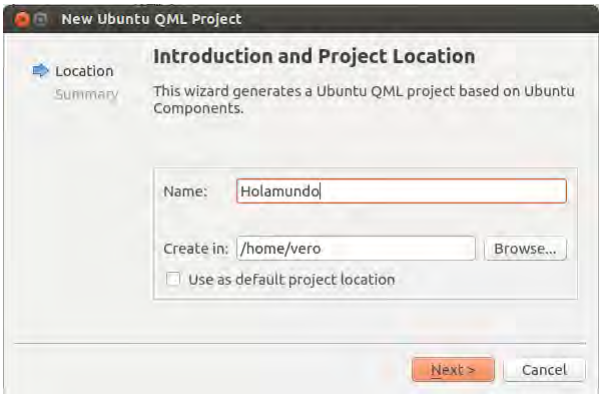


Figura F.2: Creando proyecto HolaMundo en Ubuntu Touch OS

Y se finaliza éste:

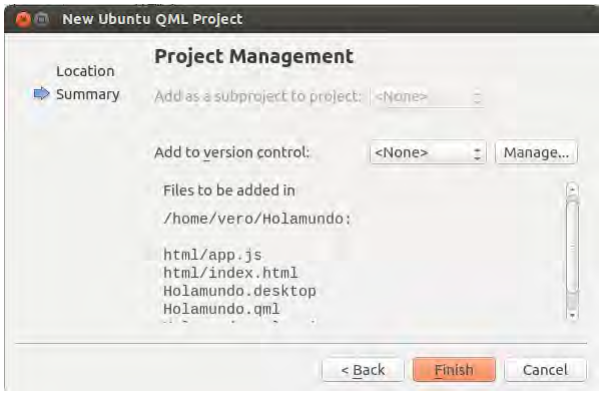


Figura F.3: Finalizando proyecto HolaMundo en Ubuntu Touch OS

Nada más finalizar la creación del proyecto, automáticamente se crean los archivos para la aplicación[1]. Una vez está creado el proyecto, ya se pueden modificar las líneas que se deseen para que se visualice el texto "Hola Mundo" en la pantalla del simulador, después, se procede a compilar y emular la aplicación ("botón que se encuentra en la barra lateral izquierda en forma de triángulo verde"):

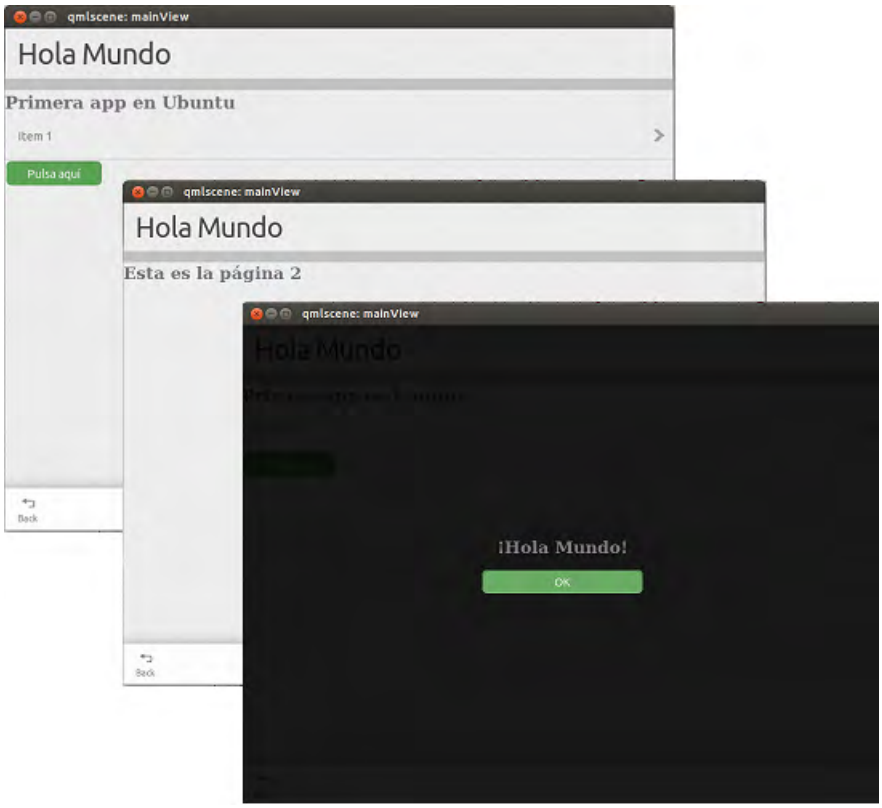


Figura F.4: Simulando aplicación HolaMundo en Ubuntu Touch OS

[1]Para conocer qué tipos de archivos se crean ir al capítulo 3.1.