Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Consumo energético en *widgets* de navegación para dispositivos móviles con pantallas táctiles

Tesis que presenta

Ana Belem Márquez Quintos

para obtener el Grado de

Maestra en Ciencias en Computación

Director de Tesis

Dr. Amilcar Meneses Viveros

México, Distrito Federal — Noviembre, 2015

A mi madre y mis hermanos

Agradecimientos

Agradezco de manera especial a mi madre Clara por su amor, su comprensión y su apoyo en todos mis sueños, ya que sin ella no hubiera sido posible que llegara hasta aquí. A mi hermano Erick por ser un ejemplo a seguir y demostrarme que no existen límites para alcanzar mis sueños. A mi hermano Edmundo por transmitir esa alegría y apoyarme incondicionalmente desde siempre.

Agradezco a mi asesor de tesis, el Doctor Amilcar Meneses, por su confianza en mí para realizar esta tesis bajo su dirección. Por su paciencia para orientarme, guiarme y por resolver mis dudas. Por todas sus enseñanzas y el conocimiento que adquirí. Por su motivación y su apoyo durante este año.

A mis sinodales, la Doctora Sonia Mendoza, la M. en C. Erika Hernández y el Doctor Francisco Rodríguez, a quienes agradezco el tiempo que se tomaron en revisar esta tesis y por sus oportunas observaciones que enriquecieron este trabajo.

A Gabriel Medina le agradezco por todo el cariño, el apoyo y los ánimos brindado durante estos dos años. También a mis amigos por su afecto y palabras de aliento en todo momento.

Al Cinvestav en especial al Departamento de Computación por la oportunidad que me brindó para continuar preparándome en una institución de alto prestigio. A las secretarías, Sofí, Felipa y Erika, por su apoyo invaluable en todos los trámites administrativos.

Finalmente agradezco a CONACyT por su apoyo económico durante estos dos años.

Resumen

Hoy en día, tenemos sistemas computacionales que utilizan demasiada energía. Los desarrolladores buscan maneras de construir sistemas computacionales con alto desempeño en cómputo y buen rendimiento de energía. Este tipo de sistemas pueden contribuir a la sustentabilidad ambiental.

Varios estudios que se han realizado, han demostrado que existe un vínculo fuerte entre la capacidad de la batería y los requerimientos de la aplicación. La principal restricción de los dispositivos móviles son las baterías. A fin de abordar este problema, los desarrolladores han implementado diferentes métodos que permiten incrementar el tiempo de uso de la batería del dispositivo mientras se mantiene un alto rendimiento. Este enfoque esta basado principalmente en la administración de algunos recursos como los procesadores y la interfaz de red.

Sin embargo estos enfoques no consideran el uso de la pantalla y el uso del *framebuffer*. Estudios anteriores muesttran que el uso de la pantalla y el *framebuffer* consumen la mayor parte de energía del dispositivo móvil. La interfaz gráfica de usuario (GUI) utiliza la pantalla y el *framebuffer*. La GUI esta compuesta de elementos gráficos para de interacción conocidos como *widgets*. Por lo tanto, los *widgets* también consumen una gran cantidad de energía en el dispositivo.

Esta tesis presenta y explica la importancia del consumo de energía para varios *widgets* de navegación. Además, este estudio considera diferentes tipos de aplicaciones, nativas y clientes del dispositivo móvil.

Abstract

In this day and age, having computer system that have higher energy is critical. Nowadays, computer engineers are looking for ways to build computer system with good computing performance and energy performance. This is kind of systems can to contribute to environmental sustainability.

Several studies that have been done, has proven there is a strong link between battery capabilities and mobile application requirements. The main constraint of mobile devices are the batteries. In order to address this problem, computer engineers have developed different methods that allow mobile device to increase the usage time of a battery while maintaining a high level of performance. This approach is based on the management of some resources such as processors and the network interface of a mobile device such as smartphones and tablets.

However, this approach does not consider the use of display and the management of a framebuffer. Previous studies show that the display and the framebuffer consume the most energy in the device. The graphical user interface (GUI) uses the display and the framebuffer. The GUI is composed of graphics elements for interaction known as widgets. Therefore, the widgets also consume high amount of energy in the device.

This thesis presents and explains the importance of energy consumption for several navigation widgets. In addition, this study considers the different types of native and client application for mobile device.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

En los últimos años, diversas áreas estudian e implementan formas de minimizar el impacto ambiental. En el área de computación se realizan técnicas para reducir el tiempo y el consumo de la energía. Este objetivo se realiza cuando los sistemas computacionales logran un buen desempeño, es decir que las tareas se completen en el menor tiempo y utilicen la menor energía posible durante su tiempo de ejecución.

Los dispositivos móviles han incrementado su uso, los más predominates son los teléfonos inteligentes y las tabletas, ver Figura 1.1. Estos dispositivos han reemplazado en gran parte a las computadoras convencionales (PC) y las computadoras portátiles, por lo que ahora son una herramienta importante para el usuario. Estos dispositivos son capaces de comunicar, entretener y realizar diversas tareas.

Existe una convergencia de las diferentes tecnologías hacia los dispositivos móviles. Estos incluyen: los reproductores de video y música, el sistema de posicionamiento global (GPS por sus siglas en inglés) y la cámara de fotografía y video, entre otros. Por tanto, algunas tareas que se ejecutan en las PC o las portátiles pueden ser realizadas por los dispositivos móviles.

Debido a su tamaño y su capacidad de movilidad, estos dispositivos presentan ventajas como lo es la portabilidad, la conectividad a diferentes redes móviles y la capacidad de realizar una variedad de tareas, entre otras. Sin embargo, estos dispositivos también presentan restricciones como la cantidad de memoria persistente y volátil (los dispositivos móviles no

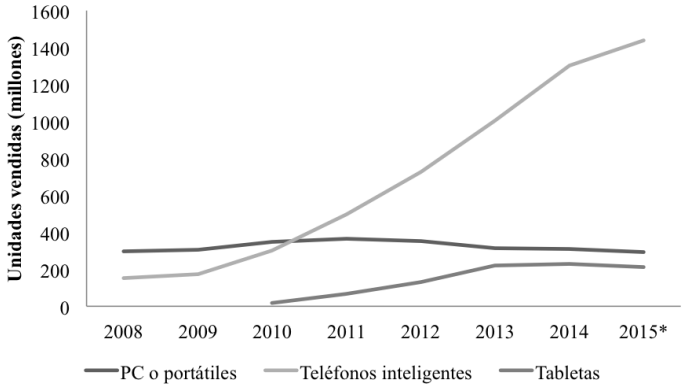


Figura 1.1. Gráfica de ventas, las ventas del 2015 es un estimado [1].

cuentan con memoria virtual), el tamaño de la pantalla, la capacidad de procesamiento y la batería. Particularmente, la capacidad de carga de la batería es la principal restricción en el dispositivo móvil.

Los servicios y aplicaciones móviles cada vez requieren de mayores capacidades de cómputo, de uso de red y de acceso a los componentes de interacción para facilitar su uso. Se ha observado una brecha de energía entre los requerimientos de la aplicación (procesamiento, despliegue de información, entre otros) y la que suministra la batería [39]. Por lo que, la capacidad de la batería es insuficiente para la exigencia de las aplicaciones móviles, ver Figura 1.2.

Existen diferentes técnicas para mejorar el rendimiento de la batería en los dispositivos móviles que permiten optimizar su uso. Estás técnicas son: 1)planificador, 2)optimización de la máquina virtual, 3) *offloading*, 4) *offlining*, 5)uso de red, 6)monitor de batería, 7)procesamiento secuencial y 8)enfoque de diseño. Las técnicas 1, 2, 3 y 7 se basan en como se ejecuta la aplicación móvil. Por ejemplo decidir como se utilizan los núcleos del procesador, en dónde se ejecuta el código de la aplicación -puede ejecutarse en el dispositivo o manera remota- o si es conveniente hacer el flujo de manera secuencial.

[1]Datos obtenidos de *International Data Corporation*

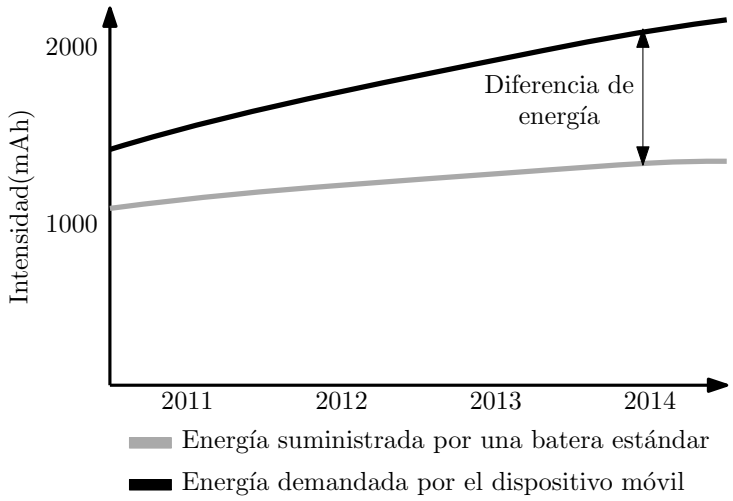


Figura 1.2. Brecha energética entre aplicaciones y batería.

La técnica *offlining* se refiere a la capacidad del cambio de estados de los componentes del dispositivo móvil (el brillo de la pantalla, el *bluetooth*, la wifi, etc.). La técnica de monitor de batería se basa de forma más directa con el usuario de acuerdo a su comportamiento y su uso. En la técnica de enfoque de diseño los estudios mencionan como se debe diseñar la interfaz gráfica de usuario (GUI) y también como reducir el número de actualizaciones de la pantalla.

1.1. Planteamiento del problema

Debido al problema de la brecha energética, las aplicaciones exigen más energía de la que puede proporcionar la batería del dispositivo móvil. Se han generado varios estudios para analizar el consumo de energía en el dispositivo móvil. Estos estudios proponen diferentes tipos de soluciones como técnicas para que la carga de la batería dure más y modelos matemáticos para estimar la energía que consume.

Por otra parte, existen diversos estudios que evalúan el consumo de energía de distintos componentes en el dispositivo móvil. Como señalan en [6, 34, 19], una gran parte del consumo de energía se debe a las comunicaciones a través de la interfaz de red (*bluetooth*, wifi , 2G, 3G, 4G, entre otros) y a la pantalla LCD (que puede ser táctil, la iluminación y el acelerador gráfico), además al estado en que se encuentre el dispositivo -estos estados pueden ser activo, incativo o suspendido-.

Para este estudio se considera que una aplicación móvil es eficiente cuando utiliza de forma adecuada los componentes del dispositivo móvil. En la literatura se han reportado diversas estrategias para lograr el ahorro de energía, como lo son: 1)planificadores a nivel de sistema operativo [50, 17, 43]; 2)optimización de máquinas virtuales que mejoran el rendimiento y la eficiencia energética [23, 33, 23]; 3)*offloading* que aumenta la capacidad de los sistemas móviles migrando la computabilidad a otros recursos computacionales [21, 22, 18, 47, 7, 15, 18]; 4)*offlining* que permite encender o apagar algún componente del dispositivo [4, 5, 6]; 5)uso de red que permite plantear estrategias para enviar y recibir paquetes [37, 31, 25, 26]; 6)monitor de batería que proporciona al usuario herramientas para monitorizar la batería y le permite tomar decisiones sobre su dispositivo [35, 13]; 7)procesamiento secuencial [36]; y 8)enfoque de diseño que a través de guías de diseño permite el manejo de algunos recursos por parte de la aplicación [4, 31, 34].

Sin embargo, el principal enfoque en estás técnicas se refiere al procesamiento y comunicación con los componentes dejando a un lado las técnicas de diseño. Los trabajos encontrados sobre las técnicas de diseño sólo se enfocan a la composición de la aplicación como lo es la selección de color, el tipo y tamaño de imágenes, lo cual es austero para el desarrollo de una aplicación y no existen estudios sobre el impacto energético de los diferentes componentes de la GUI.

Problema específico

Como se mencionó en la Sección 1.1, la pantalla es un componente del dispositivo móvil que consume mucha energía y según [6, 34, 19] consume aún más energía cuando el dispositivo se encuentra en modo activo. Este componente se encarga de desplegar la GUI, realizar el render para mostrar los datos multimedia y, en caso, de ser una pantalla táctil también se encarga de atender los eventos del usuario.

El sistema operativo (S.O.) interpreta los eventos como gestos. Estos gestos pueden variar de acuerdo al S.O. La GUI es el principal medio de comunicación que existe entre el usuario y las aplicaciones móviles. El diseño de la GUI de una aplicación establece como debe ser la navegación y como presentar los controles gráficos, los estados y la información de la aplicación al usuario. Por lo que, el enfoque de diseño se basa en optimizar estas tareas e intenta mantener la usabilidad de la aplicación. Sin embargo, como se ha visto en la literatura el dispositivo en estado activo, la pantalla consume más energía y el diseño de la GUI no toma en cuenta las repercusiones energéticas de sus componentes gráficos.

Los componentes gráficos de la GUI se le denomina *widget*. Existen diversos *widgets* que permiten al usuario navegar, controlar y recibir o visualizar información. Estos *widgets* son importantes debido a que permiten que la aplicación sea funcional. Los *widgets* de navegación son los más utilizados por las aplicaciones móviles, por que permiten navegar en las diferentes vistas [11]. Estos *widgets* de navegación son de diferentes tipos y varían de acuerdo a la plataforma móvil (dispositivo móvil y sistema operativo). Por esta razón los *widgets* de navegación son los más indispensables y utilizados en la aplicación.

Es necesario un estudio más detallado de las repercusiones energéticas de la GUI, en particular de los *widgets* de navegación. Con este estudio podemos conocer el impacto del uso de la batería al utilizar los *widgets* en las aplicaciones móviles.

1.2. Propuesta de solución

En la literatura se detectó que un dispositivo móvil en modo activo consume la mayor energía. En este estado el uso de la pantalla es el principal componente que consume energía [4]. Para disminuir el impacto en este componente se han implementado guías de diseño como el tipo de color o las características de una imagen que se debe utilizar, así como el manejo del buffer para el despliegue del contenido multimedia [19]. Sin embargo, no se ha contemplado el impacto energético del uso de los *widgets* que conforman la GUI. En particular, los *widgets* más utilizados son los de navegación. Por lo que un estudio sobre *widgets* nos dará los indicios en el impacto energético en la GUI.

Una característica principal de las aplicaciones móviles es que son fáciles e intuitivas para utilizarlas, es decir son usables. Este aspecto de usabilidad es importante y lo que buscamos es que los *widgets* de bajo consumo mantengan esta propiedad. Podemos considerar un conjunto de *widgets* tienen la misma funcionalidad y usabilidad, es decir son equivalentes [45].

Las aplicaciones móviles pueden ser nativas o *rich-Client*. Las aplicaciones nativas son aquellas que deben de ser desarrolladas y compiladas para cada plataforma. Las aplicaciones *rich-Client* son aquellas que se pueden ejecutar en un navegador web sobre cualquier plataforma, estás son capaces de reconocer los componentes y la configuración del dispositivo móvil, por ejemplo WebGL que permite mostrar gráficos en 3D en navegadores web utilizando el *hardware* del dispositivo móvil.

Este tipo de aplicaciones pueden ejecutarse sobre una máquina virtual o de manera nativa. La máquina virtual es un *software* que simula las características de una plataforma. Existen dos tipos: las máquinas virtuales de sistema que permite crear un entorno virtual con las características específicas del S.O. y las máquinas virtuales de proceso que se ejecutan cada vez que una aplicación o servicio se lanza y se detiene cuando esta se finaliza,

las más conocidas son *Java Virtual Machine* (JVM) y *Common Lenguage Runtime* (CLR)

En esta tesis abarcaremos un estudio energético entre algunos *widgets* de navegación equivalentes. Para ello trabajaremos con prototipos que evaluarán el consumo de energía. Estos prototipos serán aplicaciones que se ejecutan en un dispositivo móvil. Se contemplan tres tipos de aplicaciones: nativas que no corren en máquina virtual, nativas que corren en máquina virtual, como lo es Android que se ejecuta en la JVM y las aplicaciones *rich-Client*. La ejecución de este tipo de aplicaciones también puede incidir en el consumo de energía.

1.3. Objetivos

1.3.1. General

Realizar un estudio del impacto energético de los *widgets* de navegación equivalentes para las aplicaciones que se ejecutan en plataformas móviles y tener un indicador basado en el consumo de energía para el diseño de la GUI de una aplicación móvil.

1.3.2. Particulares

- Seleccionar los *widgets* de navegación equivalentes.
- Seleccionar una plataforma móvil (*hardware*, S.O y herramientas de desarrollo) para realizar las implementaciones de los *widgets*.
- Desarrollar los prototipos para cada *widget* y cada tipo de aplicación nativa y *rich-Client* del dispositivo móvil.
- Realizar las pruebas para las mediciones en la plataforma móvil.

- Evaluar las mediciones para determinar el consumo de energía y comparar los resultados entre las aplicaciones nativas y *Rich-client* tanto las que utilizan una máquina virtual y las que no.

1.4. Justificación

Actualmente, diversas áreas buscan el cuidado del medio ambiente. Algunas adaptan técnicas e implementan estrategias para reducir la contaminación, otras realizan tecnologías que sean autosustentables y otras reducen y re-aprovechan el consumo energético y/o la energía que utilizan. En el área de computación se busca, principalmente, el ahorro de energía al hacer procesos eficientes y de alto rendimiento, pero que optimicen la energía que están consumiendo.

Los dispositivos móviles, como teléfonos inteligentes o tabletas, son una herramienta cotidiana para el usuario, ya que son más fáciles de utilizar y son más intuitivos en el manejo de sus aplicaciones, es decir son más usables. Por tal motivo, los desarrolladores de aplicaciones se enfocan más en usuario que en el manejo de restricciones del dispositivo móvil. La batería es la principal restricción. Se han desarrollado diversas estrategias para el ahorro de energía en estos dispositivos que son proporcionadas por las plataformas móviles ya sea por *software* o *hardware*. Sin embargo, el desarrollador debe contribuir para que la aplicación sea eficiente en tiempo y energía. Actualmente, el desarrollo de aplicaciones son muchas las técnicas de *offloading* y no en el diseño de la interfaz, lo cual es paradójico ya que la pantalla consume más energía en comparación con los demás componentes.

En esta tesis se estudiará la interfaz gráfica de usuario en aplicaciones móviles de dispositivos con pantallas táctiles. En particular, nos enfocaremos en los *widgets* de navegación y mostraremos los resultados sobre el impacto en la batería que tienen al ejecutarse. Como existen diferentes plataformas móviles, en este estudio consideramos el entorno de ejecución y el *widget*

equivalente entre plataformas. Con esto obtendremos un nuevo indicador que ayude al desarrollo de aplicaciones y que permita que la carga de la batería dure más sin restar la usabilidad, es decir, que se mantengan los requerimientos de la interacción humano computadora (HCI).

1.5. Metodología

Para realizar este trabajo de tesis se realizan las siguiente tareas:

- Identificar los tipos de *widgets* de navegación para diferentes aplicaciones.
- Definir una plataforma móvil para realizar las pruebas.
- Cargar las herramientas necesarias en la plataforma móvil como una máquina virtual y navegadores que se ejecuten en tanto de manera nativa como con JVM.
- Definir el método que se utilizará para medir el consumo energético de los prototipos.
- Realizar una propuesta de diversas tareas para probar los *widgets* de navegación en las diferentes plataformas.
- Realizar los prototipos para las diferentes aplicaciones que se mencionan a continuación:
    1. Prototipo para aplicación nativa en máquina virtual.
    2. Prototipo para aplicación nativa sin máquina virtual.
    3. Prototipo para aplicación *Rich-Client* en un navegador en máquina virtual.
    4. Prototipo para aplicación *Rich-Client* en un navegador sin máquina virtual.

- Realizar las mediciones de cada prototipo y determinar su consumo energético.
- Obtener resultados, realizar las comparaciones entre aplicaciones y desarrollar las conclusiones.

1.6. Contribuciones y resultados esperados

En este trabajo de tesis se extenderá el estado del arte del consumo de energía en los dispositivos móviles. Se enfocará en la capa de presentación con el estudio de los *widgets* de navegación en diversas aplicaciones en dispositivos móviles y se realizará una comparativa entre ellos.

Se realizará la contribución al diseño de estrategias de las aplicaciones que permitan la navegación a través de ellas y que el consumo de energía sea el óptimo. Se tomará en cuenta las aplicaciones nativas y *Rich-Client* que trabajan con máquina virtual y sin ella.

1.7. Organización de la tesis

El presente documento esta formado por 6 capítulos que describen el marco teórico la propuesta de trabajo y las pruebas experimentales que se realizaron y los resultados experimentales que se obtuvieron.

El **Capítulo 2** aborda los componentes de un dispositivo móvil y se detalla el consumo de energía de cada uno de sus componentes. Además, se presentan las diferentes técnicas que existen en la literatura para el ahorro de energía en los dispositivos móviles. El **Capítulo 3** presenta la estructura de la interfaz gráfica para dispositivos móviles y los componentes que la conforman. En este capítulo se especifican los diferentes *widgets*, como se clasifican, y por ultimo, se detallan los *widgets* de navegación y la equivalencia que existe entre ellos. El **Capítulo 4** estudia los tipos de aplicaciones que existen y se especifican para dispositivos móviles. También se plantea los métodos

que existen para medir la energía consumida por un dispositivo móvil tanto en *hardware* como en *software*. El **Capítulo 5** describe los prototipos y la forma de cómo se realizaron las mediciones, se presentan los resultados con las comparaciones pertinentes y se discute la relevancia de los resultados. Finalmente, en el último capítulo se realiza una discusión sobre los resultados y se hace un análisis sobre el impacto que tienen los *widgets* de navegación en la GUI. Se dan las conclusiones del comportamiento energético de los *widgets* sobre las diferentes aplicaciones que se ejecutan en los dispositivos móviles. Por último se presentan los trabajos a futuro que se derivan en esta investigación.

CAPÍTULO 2

Dispositivos móviles

Los dispositivos móviles han evolucionado desde sus orígenes como simples dispositivos de radio comunicación hasta los dispositivos ultraligeros que realizan diversas tareas. Así mismo los componentes que lo conforman han evolucionado, permitiendo que sean portátiles, más pequeños y que se encuentren siempre conectados de forma inalámbrica. Estos dispositivos han dejado a un lado el teclado numérico para dar paso a pantallas más grandes y, por tanto, táctiles. En consecuencia, para que el usuario pueda interactuar con el dispositivo, existen diferentes eventos llamados gestos táctiles. Además, la evolución creciente de los dispositivos móviles han hecho que sean más usables, por lo cual genera que los usuarios lo vuelvan una herramienta cotidiana y por tanto, las aplicaciones requieran utilizar cada vez más los componentes del dispositivo móvil. Sin embargo, esto provoca que las aplicaciones consuman más energía. Se han realizado diversas técnicas para ahorrar energía en el dispositivo móvil que proporciona el *hardware*, el sistema operativo o el mismo desarrollador.

En la Sección 2.1 se describen los componentes de un dispositivo móvil. En la Sección 2.2 se hace una descripción de energía en los diferentes estados del dispositivo móvil. En la Sección 2.3 se explican las técnicas que existen para ahorrar energía. Por último, en la Sección 2.4 se realiza un breve resumen del capítulo resaltando los puntos más relevantes.

2.1. Componentes del dispositivo móvil

Los dispositivos móviles cuentan con características que permiten realizar diversas tareas y funciones que faciliten la movilidad del usuario. Estas tareas pueden ser: realizar llamadas telefónicas, tomar fotografías y videos, acceder a Internet y procesar información, entre otras. Los dispositivos móviles cuentan con un *hardware* específico para cumplir con dichas funciones.

En la Figura 2.1 se observa un diagrama del dispositivo móvil con los buses de comunicación entre los componentes y la alimentación de corriente suministrada por la batería. El procesador móvil es el componente central para el S.O. o las aplicaciones, utiliza bajos voltajes y puede ejecutarse en diferentes niveles de potencia, los principales en el mercado son los ARM y MIPS. La SDRAM tiene una interfaz que es controlada por una señal de reloj y se sincroniza con el bus del sistema del dispositivo móvil, puede tener un tamaño de hasta 3GB de almacenamiento. El módulo de extensión de almacenamiento se encarga de guardar datos. La interfaz de red que permite la transmisión de comunicación y datos a través de los diferentes protocolos de red (2G, 3G, 4G, wifi, *bluetooth* y GPS). La pantalla es el componente que despliega la información en datos multimedia y en el caso de ser táctil, permite que el usuario interactúe con la aplicación móvil.

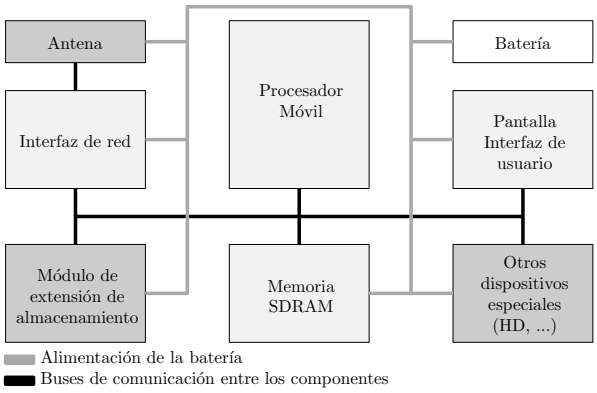


Figura 2.1. Diagrama de un dispositivo móvil.

2.2. Consumo de energía de un dispositivo móvil

En un dispositivo móvil, el consumo de energía puede variar de acuerdo al tiempo que las aplicaciones móviles utilizan los componentes. También puede variar según el estado en que se encuentre el dispositivo móvil. Estos estados pueden ser: activo, inactivo y suspendido [4, 5]. En esta sección se describen cada uno de los estados.

2.2.1. Estado suspendido

El dispositivo se encuentra suspendido cuando entra en modo de espera, es decir la pantalla se encuentra apagada, las aplicaciones se encuentran inactivas y sólo la antena está activa para recibir llamadas o mensajes. En la Figura 2.2 se muestra la potencia que requiere un dispositivo móvil en estado suspendido. Se observa el consumo de energía de los componentes principales del dispositivo móvil. Como en este estado la antena se mantiene encendida, es el componente que consume más energía.

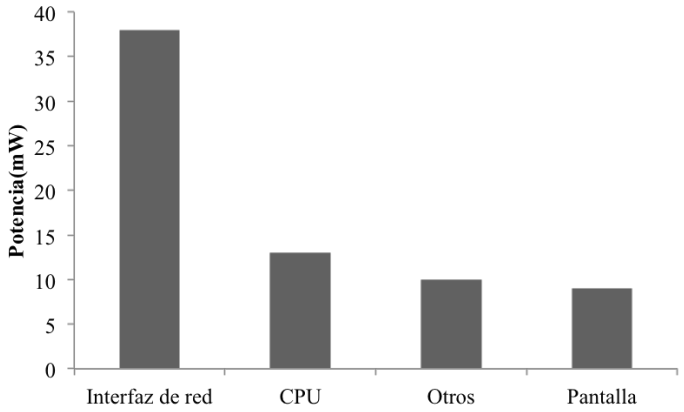


Figura 2.2. Comportamiento de la potencia en modo suspendido[1].

[1]Datos obtenidos de [4]

2.2.2. Estado inactivo

El dispositivo móvil se encuentra inactivo cuando la pantalla está encendida pero no hay aplicaciones ejecutándose en primer plano. En la Figura 2.3 se muestra la potencia que requiere un dispositivo móvil en estado inactivo. Se muestra la cantidad de energía que utiliza cada uno de los componentes del dispositivo móvil. Aunque el dispositivo móvil no se encuentra en uso, la pantalla es el componente que utiliza la mayor parte de energía.

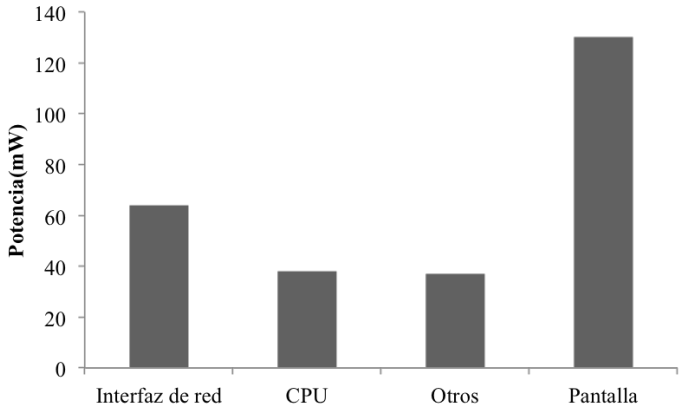
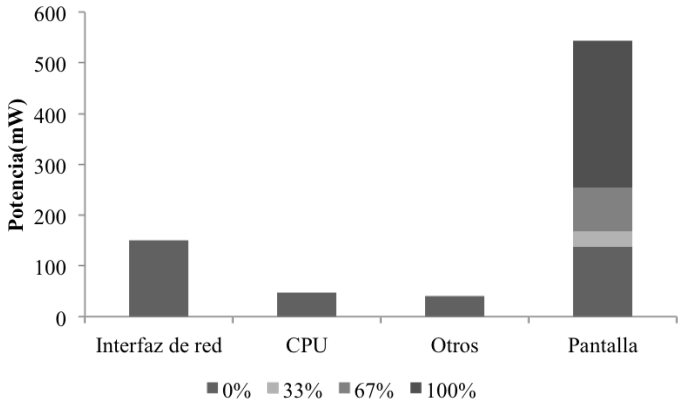


Figura 2.3. Comportamiento de la potencia en modo inactivo[2].

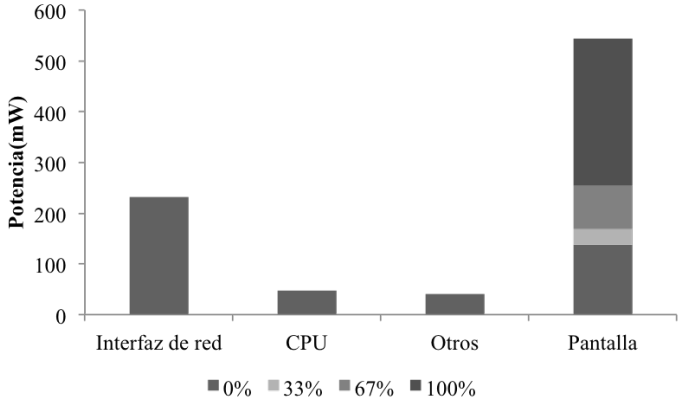
2.2.3. Estado activo

El dispositivo móvil se encuentra activo cuando hay aplicaciones ejecutándose en primer plano. Este estado es el que consume más energía. En la Figura 2.4 se ilustra la energía en estado activo de un dispositivo móvil utilizando el navegador Web con la red WiFi y con la red GPRS. En la Figura 2.4a se observa que la pantalla utiliza más potencia cuando el brillo se ajusta a más del 30 % de la iluminación y la Figura 2.4b muestra que la pantalla consume más potencia cuando el brillo es mayor al 45 % de la iluminación.

[2]Datos obtenidos de [4]



(a) Consumo de potencia utilizando la wifi.



(b) Consumo de potencia utilizando GPRS.

Figura 2.4. Comportamiento de la potencia en modo activo de un navegador Web[3].

Podemos observar que los componentes que utilizan más energía son: la antena, la pantalla y el CPU, sin importar en que estado se encuentre el dispositivo. En particular, en el estado activo la pantalla es la que consume la mayor parte de energía. Esto se debe a que es la encargada de desplegar y refrescar la información y, en caso de ser táctil, se encarga de atender los eventos del usuario.

[3]Datos obtenidos de [4]

Sin embargo, cuando se trabaja en entornos no controlados, es decir en ambientes abiertos, el consumo de energía puede ser variable. Por ejemplo cuando en el ambiente hay demasiada luz se tiende a aumentar el brillo de la pantalla para verla con mejor nitidez, esto por consiguiente eleva el consumo de energía. Por otra parte, cuando en el ambiente hay poca luz, se puede disminuir el brillo de la pantalla, por lo cual el consumo de energía disminuye. No obstante, pueden existir escenarios en un ambiente híbrido y que el comportamiento de la luz sea inestable. En este caso, algunos dispositivos móviles son capaces de controlar por sí mismos el brillo ideal. Por lo tanto, el comportamiento del consumo de energía en un escenario híbrido es muy inestable.

2.3. Técnicas de ahorro de energía

En los últimos años la prioridad de los desarrolladores de dispositivos móviles es realizar sistemas computacionales con alto rendimiento y eficiencia en procesamiento que utilicen la menor energía posible. En la literatura, se proponen diferentes métodos y estrategias que, de acuerdo al tipo de componente, tienen la finalidad de reducir el consumo de energía y mantener el rendimiento y la eficiencia en el dispositivo móvil. A continuación se presentan las técnicas que permiten dicho ahorro de energía.

2.3.1. Planificador

Esta técnica permite ahorrar energía a través del uso apropiado de los procesadores multinúcleo. La selección adecuada del núcleo que ejecuta un proceso permite un ahorro de energía cuando se trabaja con las arquitecturas multinúcleo no simétricas (un núcleo es asignado maestro que se encarga de repartir las tareas a los demás núcleos llamados esclavos) [6], dado que algunas tareas no tan importantes se puede permitir núcleos en bajo rendimiento y bajo consumo de energía. Otra estrategia es evitar la

migración de procesos entre múltiples núcleos de un procesador, por lo que el movimiento de la caché se evita y se ahorra energía.

Yuan et al. [50] presentan un planificador de energía en tiempo real eficiente, que se utiliza para ejecutar aplicaciones multimedia. Asignan estadísticamente ciclos de reloj a las aplicaciones individuales para que se ejecuten en diferentes tiempos y velocidades. Esto permite un ahorro de energía de hasta un 72 %.

Hou et al. [17] estudian los patrones que existen entre las aplicaciones y su uso. De acuerdo al uso de las aplicaciones determinan un esquema en el que deben ejecutarse y muestran un resultado de ahorro de energía de un 19 %. Por otra parte, Tseng et al. [43] gestionan los recursos del CPU basándose en los hilos en ejecución. De acuerdo al estado en que se encuentre ejecutándose la aplicación asigna los recursos y la utilización de los núcleos de una manera equilibrada.

A los dispositivos móviles, esta técnica les beneficia, ya que al contar con varios núcleos y tener recursos limitados como la memoria, permite que los núcleos sean utilizados de manera óptima y sólo cuando es necesario, permitiendo que la carga de batería dure más.

2.3.2. Optimización de la máquina virtual

Varios sistemas operativos móviles se ejecutan en máquinas virtuales. En diversos estudios [23, 33, 23] se han realizado considerables esfuerzos para el desarrollo de éstas máquinas virtuales en las plataformas móviles permitiendo una mejora en la eficiencia energética.

Un ejemplo puede ser el que presenta Kolin et al. [33] donde explican la máquina virtual Dalvik JVM que está optimizada para una CPU lenta, poca memoria RAM y corre en sistemas operativos sin memoria virtual. Otra estrategia presentada por Tapas et al. [23] es utilizar el soporte *Just In Time* para evitar el uso de la capa de la máquina virtual y también se puede combinar con el software que se encarga de traducir las instrucciones binarias

de manera dinámica para mejorar el tiempo y el consumo de energía.

2.3.3. *Offloading*

La computación *offloading* es una técnica utilizada principalmente en aplicaciones que se ejecutan en los dispositivos con recursos limitados. Estos recursos son la batería, la memoria, el procesador, el almacenamiento y las comunicaciones. La idea principal de esta técnica es migrar la ejecución de tareas computacionalmente pesadas a otros recursos fuera del dispositivos, tales como servidores [21, 22].

Para lograr un uso apropiado de esta técnica es importante identificar la tarea que tiene las características para ejecutarse fuera del dispositivo. Después se calcula, en función del tiempo y energía, lo que costaría enviar la tarea para que se ejecute fuera del dispositivo tales como transferencia de datos, código de transferencia, servicios de aplicaciones y el uso de la antena, entre otros. También, en algunos casos, puede considerarse el uso de memoria como una referencia para ver si una tarea está sujeta a ser enviada para funcionar fuera del dispositivo. Algunos tipos de tareas que están sujetas a estás técnicas son reconocimiento de voz o de procesamiento de imagen [18]. Los autores de [2, 9, 21, 22, 38, 30] se basan en esta técnica.

La técnica de *offloading* puede ayudar a mejorar el rendimiento y el ahorro de energía [22]. En este trabajo nos concentraremos en el ahorro de energía. En [22] los autores proponen una desigualdad para tomar la decisión de si un conjunto de instrucciones se ejecutará de manera *offloading* o en el dispositivo. Suponemos que C son la cantidad de instrucciones que tienen que ejecutar, S y M es la velocidad de instrucciones por segundo que puede procesar el servidor y el dispositivo móvil respectivamente. D representa la cantidad de intercambio de *bytes* entre el servidor y el dispositivo y B es el ancho de banda disponible. P_c es la potencia eléctrica al ejecutar C instrucciones en el dispositivo, P_i es la potencia eléctrica mientras el dispositivo se encuentra inactivo y P_{tr} es la potencia eléctrica que se necesita

para enviar y recibir datos.

La energía que se necesita para ejecutar las C instrucciones en el dispositivo móvil se muestra en la Ecuación 2.1, mientras que la energía necesaria al realizar *offloading* se muestra en la Ecuación 2.2. Para exista un ahorro de energía en el dispositivo móvil la Ecuación 2.2 debe ser menor que la Ecuación 2.1,

$$P_c \times \frac{C}{M} \tag{2.1}$$

$$P_i \times \frac{C}{S} + P_{tr} \times \frac{D}{B}. \tag{2.2}$$

Cuervo et al. [7] presentan un sistema que soporta código *offload* de granularidad fina para maximizar el ahorro de energía con el mínimo esfuerzo del desarrollador. Este sistema decide, en tiempo de ejecución, qué métodos deben ser procesados de forma remota para lograr un mayor ahorro de energía de acuerdo a las restricciones sobre la conectividad móvil.

Hong et al. [15] desarrollan un sistema de recuperación de imágenes basado en el contenido, el cual realiza un cálculo intenso para la búsqueda de imágenes. En este trabajo, se decide en tiempo de ejecución qué es lo que debe ser enviado al servidor. Esta decisión depende del ancho de banda de la red inalámbrica, el tamaño de todas las imágenes necesarias y el número búsquedas realizadas por el usuario. Los resultados muestran un ahorro de hasta 45 %. Otro ejemplo de esta técnica se presenta en [18] donde se calcula mediante un grafo la energía que consumirá el dispositivo móvil y decide en donde se ejecutará la función, a través de un algoritmo que obtiene una solución óptima para el ahorro de energía en el dispositivo móvil.

Esa técnica ahorra energía significativa en el dispositivo móvil y mejora su rendimiento. Sin embargo, presenta algunas desventajas. Puede existir un alto consumo de energía hacia donde es migrada la computabilidad y por lo tanto genera más contaminación. Por otro lado, los desarrolladores practican esta estrategia para que su aplicación sea eficiente en tiempo y energía, sin

embargo muchas veces esta técnica depende de la calidad del ancho de banda.

2.3.4. *Offlining*

Esta técnica se refiere al cambio de estado de un componente en un dispositivo móvil. Por ejemplo, el cambio de estado de algunos protocolos de la interfaz de red, como el wifi y la 3G cuando el porcentaje de la batería se encuentre bajo. También el apagar la pantalla después de cierto tiempo de inactividad del usuario con el dispositivo móvil.

En algunos trabajos los autores estudian el consumo energético de un dispositivo móvil en sus diferentes estados: activo, inactivo y suspendido [4, 5]. Aquí se muestra que al mantener el estado suspendido se consume menos energía porque sólo quedan activos los componentes necesarios como la antena y se ejecutan sólo los servicios requeridos.

Carroll et al. [6] proponen políticas de acuerdo con el estado del procesador, es decir, establecen si el número de subprocesos que se ejecutan excede la capacidad del núcleo, a continuación, se enciendan núcleos adicionales (si los hay), por otro lado, si los núcleos no están en uso estos se apagan.

2.3.5. Uso de red

El uso de red se basa en la transmisión de comunicación y datos en las diferentes redes móviles. Principalmente la estrategia que se aplica es la de utilizar la relación de cola para ahorrar energía. La relación de cola es la energía que se utiliza después de descargar un paquete [37]. En la Figura 2.5 se muestra un ejemplo de la energía de la cola de red. En esta, la comunicación de los datos termina en $7s$ pero se mantiene en un estado de energía durante $10s$ después de terminada la comunicación.

Mital et al. [31] muestran que no importa el tamaño del archivo de datos para descargar, ya que la energía está dominada por el estado de la

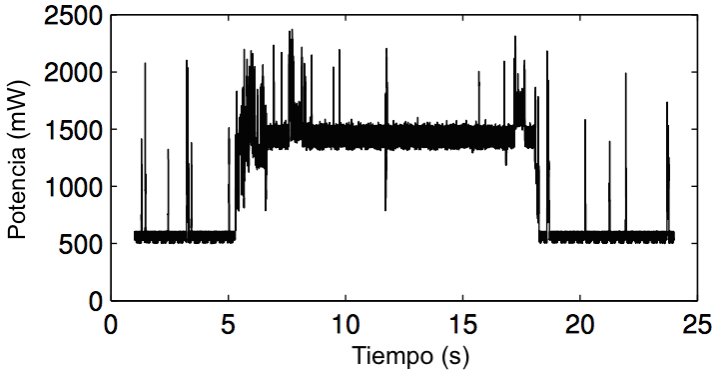


Figura 2.5. Medición de energía de la cola de red [37].

cola o de radio en redes celulares. Liu et al. [25] realizan un algoritmo de doble cola con la finalidad de programar otras transmisiones a la cola. Este método puede reducir el retardo y la transición de sobrecarga que se produce entre diferentes estados debido a predicciones incorrectas. Sin embargo, este método está limitado con base en los escenarios de pequeñas transmisiones. Por otra parte, Looga et al. [26] proponen un planificador para el tráfico en las redes inalámbricas que realiza retrasos de comunicación cortas y los combina para reducir el tiempo y aprovechar las colas de radio que se generaron. Este planificador puede ahorrar hasta un 72 % de la energía consumida por el dispositivo.

2.3.6. Monitor de la batería

Esta técnica implica la interacción humano-computadora. Algunos autores la llaman técnica de la interacción humano-batería [35]. Esta permite a los usuarios monitorear el uso de la batería y gestionen las aplicaciones de su dispositivo móvil para optimizar la batería.

Heikkinen et al. [13] llevaron a cabo un estudio sobre las actitudes y el comportamiento de los usuarios de dispositivos móviles. Observaron que los usuarios tienden a ajustar los distintos componentes que ofrece el dispositivo,

tales como el brillo de la pantalla, la red inalámbrica y *bluetooth*, entre otros componentes. Sin embargo, los resultados hacen hincapié que no es suficiente sólo realizar ajustes, sino también exigen información sobre el estado de la batería más clara y detallada, así como las aplicaciones que se encuentran consumiendo demasiada energía. Por otro lado, los usuarios también quieren entender cómo las diferentes aplicaciones y servicios afectan el consumo de energía de su dispositivo con la finalidad de que aprendan a controlarlo.

2.3.7. Procesamiento secuencial

El uso de procesadores multinúcleos en dispositivos móviles permiten ejecutar código en paralelo o incorporar múltiples hilos a las aplicaciones móviles. Sin embargo, la mayor parte del código de las aplicaciones móviles son en serie. En el trabajo [36] los autores muestran que el uso de procesadores multinúcleos afectan la eficiencia energética en la ejecución de tareas en paralelo, este comportamiento se debe a la arquitectura de la alimentación de energía con núcleos en el chip. En particular, los procesadores multinúcleos para dispositivos móviles se comportan como tipo *offlining*, es decir, los núcleos se encienden o se apagan dependiendo de su uso. Esto significa que la energía aumenta a medida que se estén ejecutando más núcleos. En [36] se muestra que el límite inferior de energía que puede tener un proceso en este tipo de procesadores multinúcleos es cuando la tarea es secuencial, es decir que la ejecución de programas paralelos consume más energía que un programa secuencial en un dispositivo móvil. En la Figura 2.6 se observa que el menor tiempo de ejecución se obtiene con aplicaciones paralelas, independientemente de si se utiliza *Java* o *Java Native Interface* (JNI que permite que un programa escrito en *Java* pueda intercatuar con otros lenguages). La Figura 2.7 muestra que ambos programas secuenciales (Java y JNI) consumen menos energía en comparación con las versiones paralelas del mismo programa.

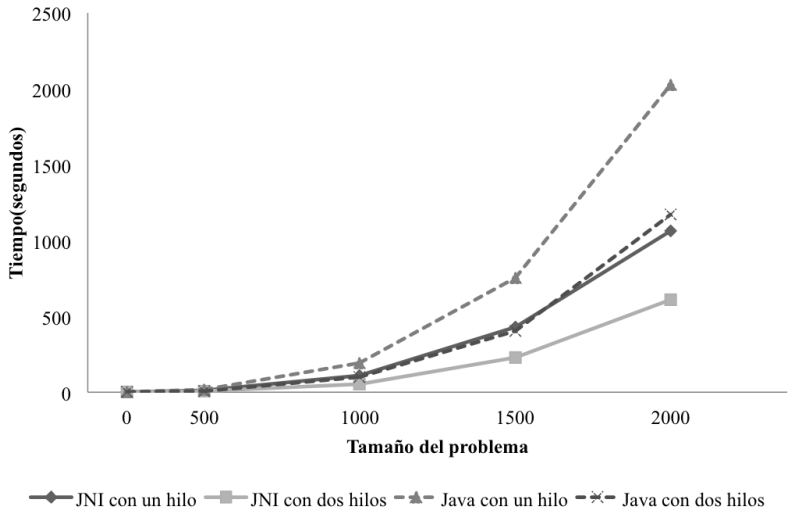


Figura 2.6. Tiempo obtenido con Linpack, utilizando Java y JNI [36].

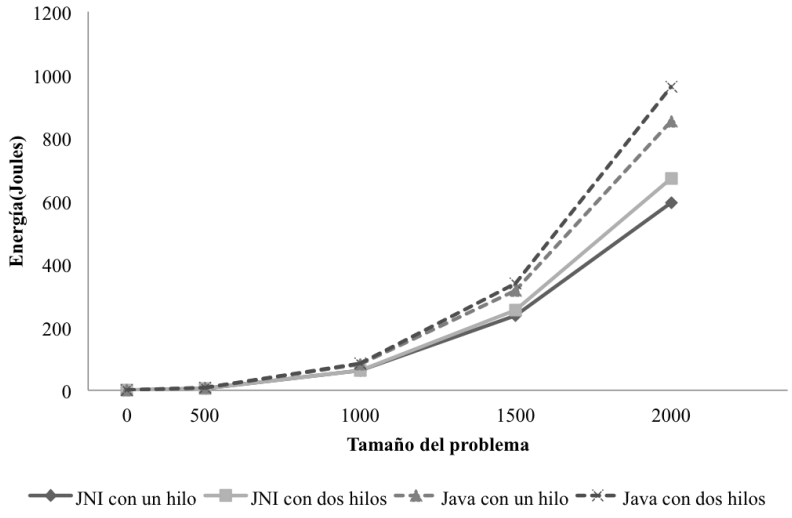


Figura 2.7. Energía con Linpack, utilizando Java y JNI [36].

Por tanto, cuando el número de hilos incrementa en un programa en paralelo, existe una reducción en tiempo de ejecución, mientras que el consumo de energía aumenta en el mismo tiempo. Entonces, para ahorrar energía en los dispositivos móviles es necesario diseñar programas que se ejecuten de manera secuencial.

2.3.8. Enfoque de diseño

Esta es una técnica de interacción humano-computadora. Permite ahorrar energía a través del estudio de la capa de presentación. Esto se refiere a la estructura de la presentación de la aplicación por ejemplo tipo de contenido a desplegar y colores.

Varias investigaciones [4, 31] coinciden en que la capa de presentación tiene un alto consumo de energía mientras se encuentra en uso. Este consumo depende principalmente del color que predomine en la pantalla, es decir, los colores más brillantes aumentan el consumo de energía [31]. Otro elemento que influye en el comportamiento de la energía en la pantalla es el nivel de brillo [4], ver Figura 2.8. Perruci et al. [34] realizan un estudio con diferentes tipos de imágenes. Los resultados muestran que el formato PNG (por sus siglas en inglés de Gráficas de Red Portátiles) puede reducir el consumo de energía significativamente.

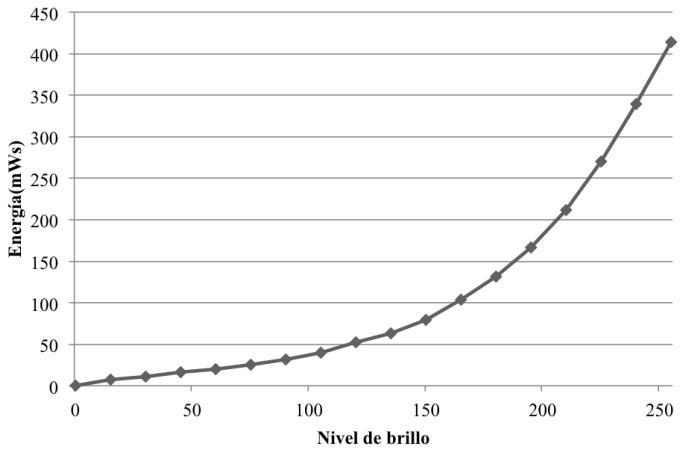


Figura 2.8. Consumo de energía en la pantalla variando el nivel de brillo.

2.4. Resumen

Tal como se presenta, existen diferentes técnicas aplicadas a los dispositivos móviles para ahorrar energía. Para revisar cómo estas estrategias afectan a un dispositivo móvil se deben considerar los principales componentes de *hardware*, tales como: pantalla, render, CPU y antena. La Tabla 2.1 permite observar el impacto de la técnica para cada componente.

Podemos notar que el enfoque *Offlining* y monitor de la batería podría tener un mayor impacto en el dispositivo móvil en general. Sin embargo, el enfoque de diseño afecta a la pantalla, el sistema de *render* y el procesamiento. Consideramos que este enfoque en este trabajo, ya que estos componentes son los principales consumidores de energía del dispositivo móvil en modo activo [4].

Técnicas	Pantalla	Render	CPU	Antena	Proporcionado
Planificador			✓		S.O móvil
Optimización de máquinas virtuales			✓		S.O móvil
Offloading			✓	✓	Desarrollador
Offlinig	✓	✓	✓	✓	S.O y HW móvil
Uso de red				✓	S.O móvil
Monitor de la batería	✓	✓	✓	✓	S.O móvil
Procesamiento secuencial			✓		Desarrollador
Enfoque de diseño	✓	✓	✓		Desarrollador

Tabla 2.1. Técnicas de ahorro de energía y su relación con los componentes de un dispositivo móvil.

CAPÍTULO 3

Interfaz gráfica de usuario para dispositivos móviles

El éxito de los dispositivos móviles es que sus aplicaciones son usables, es decir fáciles e intuitivas de utilizar [27]. Esto implica que los componentes de la interfaz gráfica de usuario cuentan con un buen diseño de interacción humano computadora. Sin embargo, cada plataforma tiene sus propias guías y patrones de diseño.

En este capítulo, se detallará la interfaz gráfica de usuario y como el usuario interactúa con la aplicación. En la Sección 3.1 se explica brevemente que es la interfaz gráfica de usuario y cómo debe ser el diseño centrado en HCI. En la Sección 3.2 se estudia como es el diseño de las aplicaciones móviles dependiendo de su entorno de programación y que diseños se han realizado para ahorrar energía. En la Sección 3.3 se aborda como es la interacción con el usuario y la interfaz gráfica. En la Sección 3.4 se aborda los componentes gráficos de la interfaz gráfica de usuario llamados *widgets*, aquí se describe su taxonomía y como un conjunto de *widgets* son equivalentes. Por último, en la Sección 3.5 se realiza un breve resumen con los puntos más sobresalientes del capítulo.

3.1. Interfaz Gráfica de Usuario

La Interfaz Gráfica de Usuario (GUI por sus siglas en inglés) es un componente que permite al usuario interactuar con las aplicaciones de un sistema. La GUI está compuesta por diferentes elementos, como son: ventanas, menús, iconos, y otros elementos visuales que representan información u otros tipos de controles. En sistemas tradicionales, la GUI se controla a través de dispositivos señaladores. Estos dispositivos señaladores pueden ser: el ratón, la bola de seguimiento, el lápiz o el dedo. En el caso de dispositivos móviles la interacción es a través de pantallas táctiles o un control señalador[1].

El diseño de la GUI es una rama del área de estudio de Interacción Humano Computadora (HCI). HCI toma en consideración varios factores del usuario para determinar el diseño de la GUI [12]: qué es lo que quiere y que espera del producto, cuáles son las habilidades y limitaciones, cómo percibe y procesa la información y qué encuentra atractivo y agradable. Por otra parte, las características técnicas y las limitaciones del *hardware* y *software* que posee el dispositivo también deben de ser consideradas para el diseño. Por lo tanto, la interfaz gráfica de usuario es la parte de la aplicación que permite que permite al usuario oír, tocar, hablar o realizar cualquier otra forma de interacción[11].

Un buen diseño de la GUI permite al usuario concentrarse en la información y las tareas que lo ocupan y no en los mecanismos para presentar la información y los procesos internos para realizar las tareas [11]. Esta GUI guía al usuario de manera intuitiva y facilitan el desplazamiento por la aplicación. Por consiguiente, la GUI debe proporcionar de forma funcional y estética los siguientes componentes [14]:

[1]El control señalador es una palanca de mando utilizado como dispositivo señalador como un panel táctil

- **Metáforas:** son los conceptos esenciales para comunicar a través de términos e imágenes.
- **Modelo mental:** se refiere a la organización de datos, funciones, tareas y roles.
- **Navegación del modelo mental:** se refiere a menús, iconos, cuadros de diálogo y ventanas emergentes.
- **Apariencia:** se refiere a la presentación visual, auditiva y verbal.
- **Interacción:** es el comportamiento de los controles interactivos de pantalla así como la entrada y salida de la información.

El éxito de un buen diseño de la GUI es que el usuario siempre tenga una retroalimentación rápida de la acción que realiza, por ejemplo si el usuario presiona un botón se observe que existe la acción de presionar o si el usuario solicita información la aplicación debe retroalimentar en que estado se encuentra (buscando, procesando, etc.).

3.2. Diseño de la GUI para aplicaciones móviles

El diseño de la GUI de cualquier aplicación siempre debe considerar los principios de HCI, es decir que sea usable. Las aplicaciones móviles no utilizan un manejo de ventanas, es decir que hay una sola ventana por aplicación. Por tanto, a través de diferentes acciones que se le permite al usuario realizar se hace un refresco de la pantalla para actualizar el contedido de la ventana. Estás acciones permiten la navegación a través de la aplicación, es decir son los *widgets* de navegación.

Dado que cada plataforma tiene sus propios patrones de diseño, los elementos y el flujo de la aplicación puede variar. En esta sección abordaremos como se realiza el diseño en las plataformas móviles (más populares) de IOS y de Android, así como las aplicaciones basadas en Web.

3.2.1. Diseño de la GUI en iOS

El diseño de la GUI en la plataforma de iOS se realiza a través del *storyboard* que es una representación visual de la apariencia y de como es el flujo de la aplicación. El *storyboard* permite ver las diferentes interfaces a través de las acciones que estén implementadas. En la Figura 3.1 se observa un prototipo de como es la navegación en la aplicación al oprimir cada botón para cambiar el color de la pantalla. Podemos considerar que los cambios de pantalla son los cambios del estado de la aplicación.

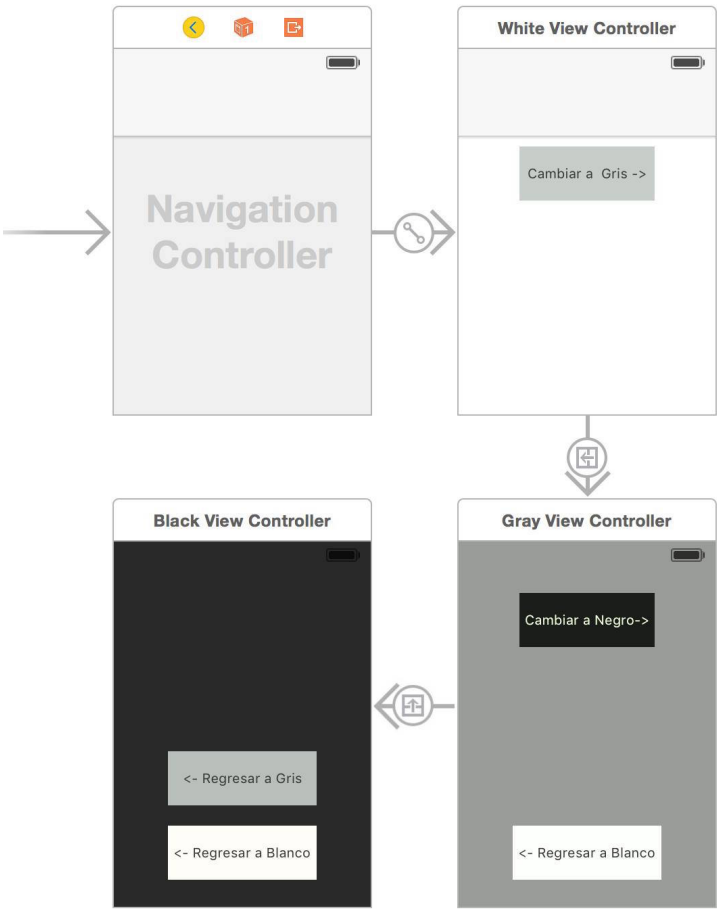


Figura 3.1. Modelo de navegación de una aplicación diseñada para iOS.

El diseño de iOS permite ajustar el contenido de la aplicación según sea la orientación del dispositivo móvil (retrato o paisaje) y el tipo de dispositivo. El desarrollador debe realizar el diseño de la GUI una sola vez y habilitar el modo de ajuste de contenido.

3.2.2. Diseño de la GUI en Android

El diseño de la GUI en la plataforma de Android se realiza a través de un conjunto de clases que se encuentran asociadas a actividades. Las vista de cada aplicación de Android esta asociada a objetos de la clase *Activity*[2] y los cambios entre las vistas se denominan *Intent*[3]. Los objetos de *Activity* controla las vistas de una ventana. Los objetos de *Intent* se encuentran asociados a algún componente de navegación. En la Figura 3.2 se observa un prototipo como es la navegación en la aplicación al oprimir cada botón para cambiar el color de la pantalla.

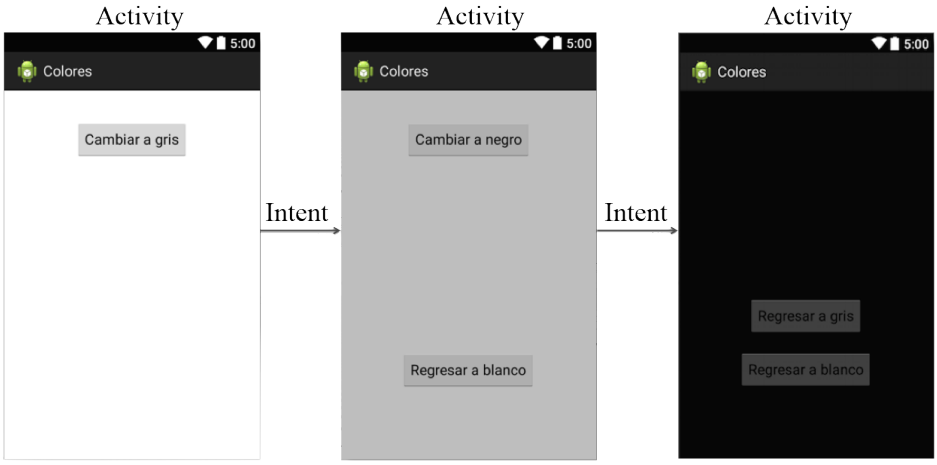


Figura 3.2. Modelo de navegación de una aplicación diseñada para Android.

[2] *Activity* es el nombre de la clase
[3] *Intent* es el nombre de la clase

El diseño de la GUI en Android también permite ajustar el contenido. Este ajuste lo realiza el desarrollador de manera manual para cada tamaño de pantalla en que se va a utilizar la aplicación y para cada cambio de orientación (retrato o paisaje).

3.2.3. Diseño de la GUI basada en Web

El diseño de la GUI basada en Web se realiza a través de un mapa de navegación que permite representar el flujo de la aplicación. Este mapa de navegación son las acciones que puede realizar el usuario. En la Figura 3.3 se observa un prototipo de como es la navegación en la aplicación al oprimir cada botón para cambiar el color de la pantalla.

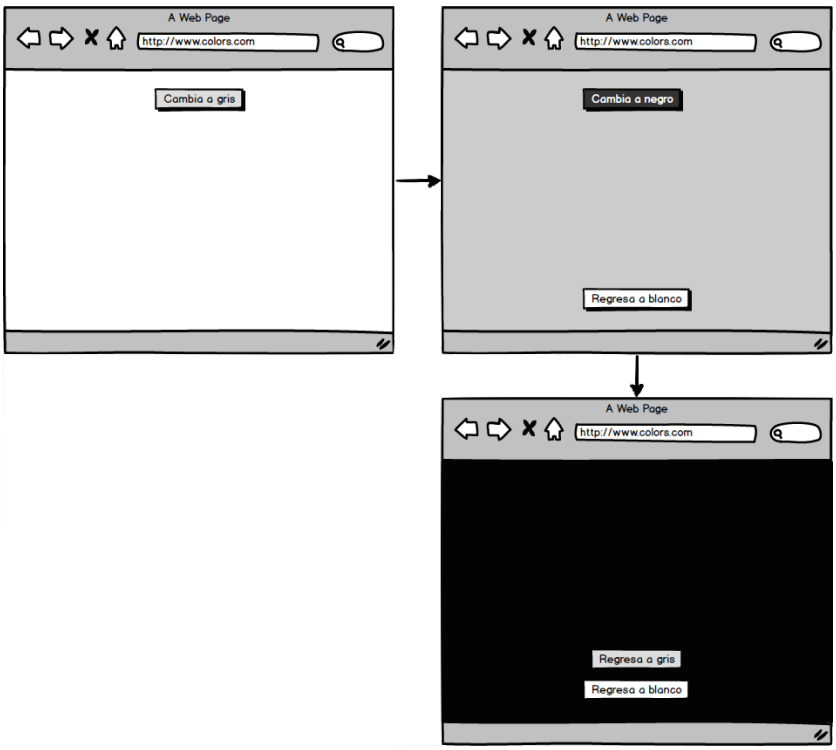


Figura 3.3. Modelo de navegación de una aplicación deseñada para la web.

El diseño web también permite adaptar las páginas al tamaño, la resolución y orientación de la pantalla, sin la necesidad de tener que configurar un dominio o subdominio específico para los dispositivos móviles. Esta adaptación permite mantener experiencias similares en diferentes tamaños de los dispositivos [11]. Para lograr esta adaptación se utilizan instrucciones en las etiquetas de la cabecera de las páginas HTML5 y consultas de CSS3 que especifican los estilos y tamaños de los elementos de la página web dependiendo del tamaño de las pantallas que lo visualizan. En la Figura 3.4 se muestra el diseño de una página web visto en diferente tamaño y orientación.

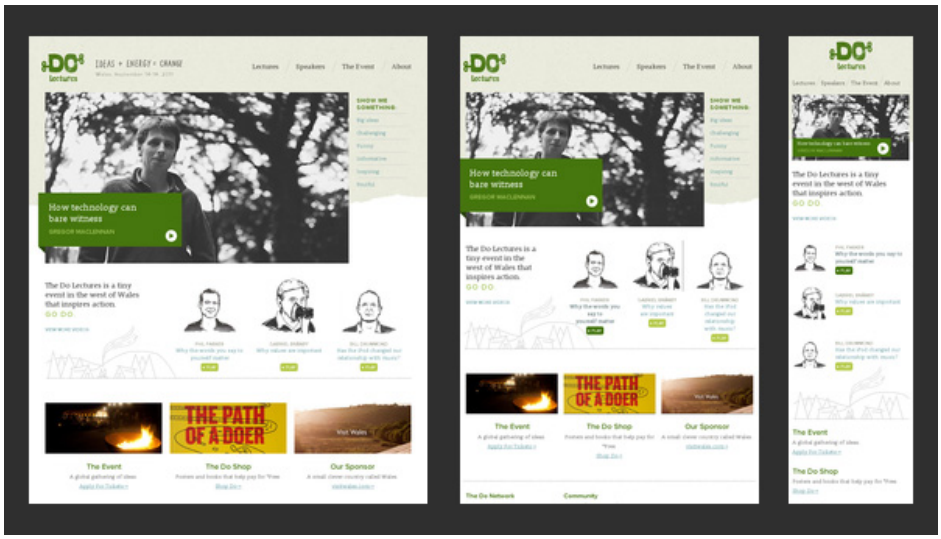


Figura 3.4. Ejemplo de un sitio web adaptable.

3.2.4. Diseño de la GUI para ahorrar energía

En la literatura se han implementado varias estrategias que ayudan a reducir el consumo de energía. Otras estrategias son una guías de estilos y colores, de tipos de datos para optimimizar el *framebuffer*[4]. Algunas de estas estrategias involucran al usuario como: representar la batería mediante un icono que permite monitorear su estado y consumo de energía, también existen aplicaciones que permiten conocer como las diferentes tareas y procesos consumen la energía o la batería.

Algunas de estas estrategias involucran al usuario como: representar la batería mediante un icono que permite monitorear su estado y consumo de energía, también existen aplicaciones que permiten conocer como las diferentes tareas y procesos consumen la energía o la batería tales como *Power Tutor*. *Power Tutor* es una herramienta para Android que es capaz de estimar la energía que esta consumiendo cada aplicación en cada componente [51]. Para IOS el sistema provee una aplicación que muestra el porcentaje de uso de la batería de cada tarea. Este tipo de aplicaciones ayudan a estimar cuanto tiempo puede durar la carga de la batería de acuerdo a las tareas que se encuentren en ejecución o al estado del dispositivo móvil.

Por parte del diseño de la GUI, por un lado se recomiendan una guía de uso colores. Los colores entre más brillantes consumen más batería [4]. Esto ha sido aprovechado por algunos autores para proponer aplicaciones que permitan bajar de manera automática el brillo y cambiar el color a tonos más oscuros a áreas de la pantalla que para el usuario no es de interés, por ejemplo *Dark Windows* [28]. En otros estudios se recomienda que, además de no utilizar colores muy brillantes en el diseño de la aplicación se aconseja que se debe cuidar el tipo de imágenes que se manejan. Estás imágenes deben de ser de formato PNG, debido al tamaño de su compresión [34].

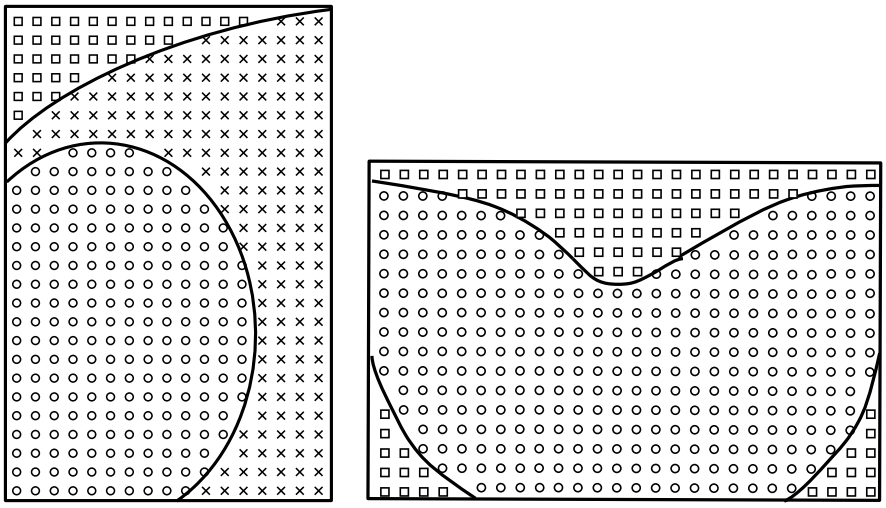
Por otra parte, se sabe que el manejo del *framebuffer* puede consumir mucha energía durante la operación de barrido de pantalla [46]. En la

[4]Número de actualizaciones que se realiza en la pantalla

estrategia del manejo del *framebuffer* se han propuesto algoritmos de compresión gradual adaptativo basado en la codificación por longitud de ejecución[5] (RLE). El algoritmo reduce la cantidad de *framebuffer* y muestra un ahorro de energía de hasta un 15 % [46].

3.3. Gestos táctiles

La mayoría de los dispositivos móviles como teléfonos inteligentes y tabletas tienden a utilizar pantallas táctiles. Por lo tanto el tipo de interacción que más se realiza entre el usuario y la aplicación es mediante la pantalla táctil a través de los diferentes eventos de usuario. Estos eventos se le llaman gestos táctiles. Es decir, a través de los gestos táctiles el usuario interacciona con una la aplicación, permitiéndole realizar las acciones para la cual la aplicación está diseñada.



(a) En modo retrato **(b)** En modo paisaje

Figura 3.5. Áreas de contacto de un dispositivo móvil [48].

[5]RLE es una forma simple de compresión de datos

Al utilizar un dispositivo móvil las áreas de interacción sobre la pantalla varían de acuerdo en como el usuario utiliza el dispositivo[11] o en la dimensiones de este. Por ejemplo, en la Figura 3.5 se muestran las áreas de interacción sobre la pantalla de una dispositivo móvil tradicional con una pantalla de cuatro pulgadas. En la 3.5a se ejemplifica la pantalla en modo retrato para usuarios diestros utilizando el dedo pulgar [48]. En la 3.5b se ejemplifica la pantalla en modo paisaje para usuarios que utilizan los dedos pulgares. Las zonas de círculos representan las zonas que se pueden llegar con mayor facilidad y la zona de cuadros es la que requiere de un mayor esfuerzo para alcanzarla, por tanto se recomienda que el diseño de los *widgets* de navegación deben de colocarse en las zonas más fáciles de alcanzar y en las que son menos alcanzables se recomienda una barra de estados de la aplicación.

Existen un conjunto de gestos táctiles que son comunes en la mayoría de las plataformas móviles. Estos gestos son la base de como se espera que los usuarios interaccionen con el dispositivo móvil y es el sistema operativo quien se encarga de interpretarlos. Esto es muy útil para los desarrolladores puedan determinar que tipos de gestos utilizar para cualquier aplicación independientemente de la plataforma. Estos pueden ser: tocar, doble toque, arrastrar y deslizar, entre otros. Por ejemplo, en la Figura 3.6 presenta los gestos básicos que describe Wroblewski [48]. Sin embargo en plataformas como Android se puede cambiar el significado de los gestos, lo cual genera un problema en el diseño de aplicaciones móviles.

3.4. Widgets

Los *widgets* son componentes gráficos de la GUI que pueden por un lado mostrar datos de diferentes tipos (imágenes, texto y video, entre otros) y por otra parte permiten al usuario interaccionar con la aplicación. Los *widgets* son elementos altamente reutilizables y se utilizan en diversas ocasiones, a través

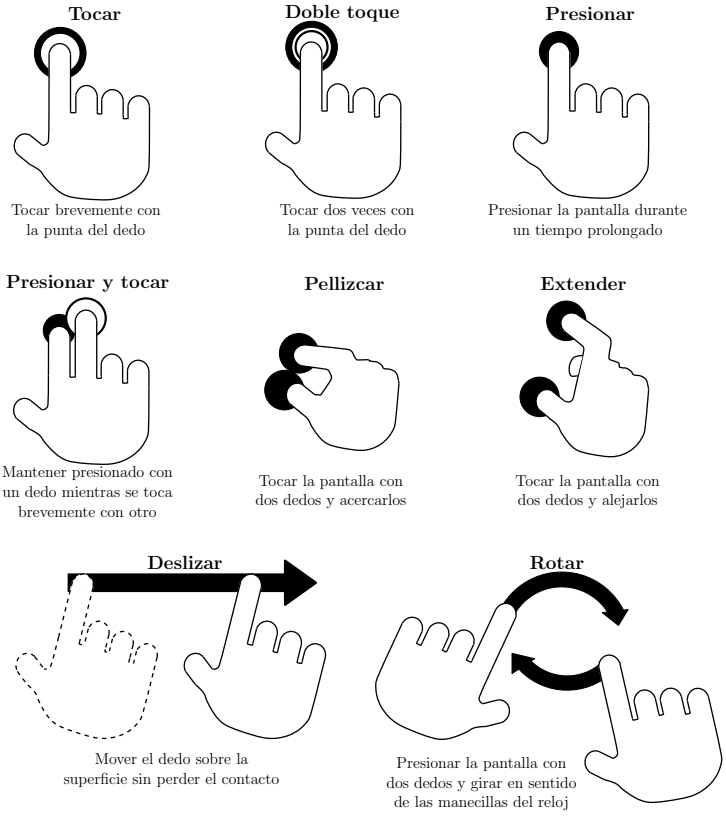


Figura 3.6. Gestos táctiles básicos.

del sistema operativo y las aplicaciones del dispositivo. Los *widgets* pueden ser utilizados para acceder rápidamente a los niveles relacionados de información, proporcionar señales visuales sobre el estado actual del dispositivo y controlar la cantidad de información y el nivel de detalle necesario en una página. Estos se pueden visualizar como botones, enlaces, iconos, indicadores o información sobre alguna herramienta [16].

Existen *widgets* simples y compuestos. Un *widget* simple es un elemento gráfico que sólo esta compuesto por un elemento simple como un campo de texto o un botón. Un *widget* compuesto consiste en dos o más *widgets* como tablas o menús.

Además, los *widgets* no son universales, debido a que cada plataforma como Android o IOS, presentan sus propios tipos. Para esto, es necesario realizar una equivalencia de *widgets*. En [45] los autores realizan una equivalencia de *widgets* tomando en cuenta la usabilidad y la funcionalidad.

3.4.1. Clasificación de Widgets

La clasificación de los *widgets* para aplicaciones móviles se describe en [16], el cual toma como base los patrones de composición, navegación y visualización de la información. La Figura 3.7 presenta una taxonomía de *widgets* con base en su funcionalidad.

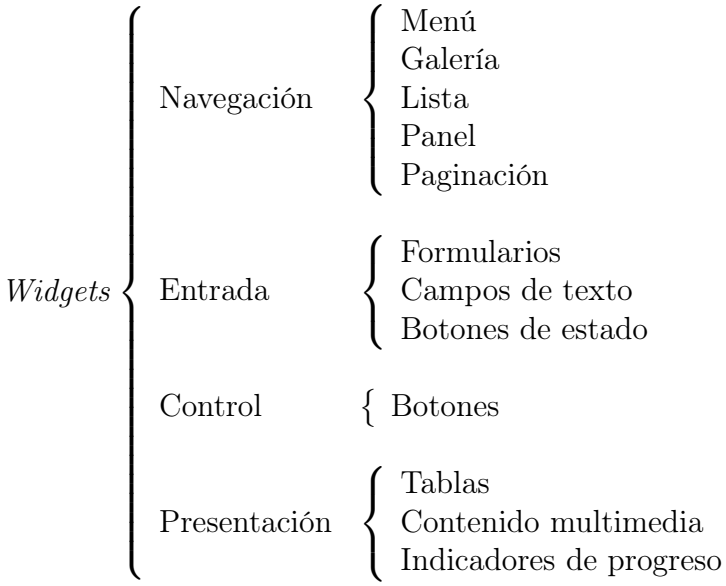


Figura 3.7. Clasificación de los widgets.

3.4.2. Equivalencia de Widgets

Como se ha comentado, los *widgets* no son universales, es decir no todos los *widgets* están disponibles en todas las plataformas móviles. Por lo que si se desea tener aplicaciones que corran en diferentes plataformas se debe asegurar que los *widgets* mantengan la funcionalidad y usabilidad entre las plataformas móviles o la orientación del dispositivo (retrato o paisaje). Para ello se requiere establecer una relación de equivalencia, lo que permite a los desarrolladores escoger que *widgets* van a utilizar entre plataformas y orientación de la ventana, manteniendo la usabilidad de su aplicación.

En [11] propone una relación de equivalencia en base a la funcionalidad y usabilidad. La funcionalidad es representada por la clasificación descrita en la sección anterior, mientras que la usabilidad es de acuerdo a pruebas con usuarios que realizó aplicando diferentes tipos de encuestas. Esta relación de equivalencia cumple que:

- **Reflexividad:** todo *widget* a en el conjunto K es equivalente consigo mismo en funcionalidad y usabilidad.
- **Simetría:** dado dos *widgets* (a, b) del conjunto K se tiene que a es equivalente a b en funcionalidad y usabilidad, entonces b es equivalente a a en funcionalidad y usabilidad.
- **Transitividad:** Dado tres *widgets* (a, b, c) del conjunto K se tiene que a es equivalente a b en funcionalidad y usabilidad y este a su vez es equivalente a c en funcionalidad y usabilidad, a es equivalente a c en funcionalidad y usabilidad.

3.5. Resumen

Como se presenta en este capítulo, la GUI es un elemento principal de una aplicación móvil. Esto se debe a que es el medio por el cual el usuario se puede comunicarse con la aplicación. Una aplicación móvil debe cumplir con los requerimientos de HCI para que sea fácil e intuitiva de utilizar.

En el diseño de la GUI son pocos los trabajos que se han enfocado en el ahorro de energía, como los que se presentan en [28, 34, 46]. Sin embargo, el manejo de la GUI es uno de los componentes de la aplicación que consume la mayor parte de la energía en una plataforma móvil.

El diseño de navegación son las acciones que representan los diferentes estados que puede tener una aplicación, es decir son las diferentes interfaces que se pueden mostrar. Sin importar el entorno de programación que se este trabajando (IOS, Android o HTML5), estas acciones son representadas por los *widgets* de navegación que permiten navegar a través de la aplicación. Por tanto, los *wigdgets* de navegación se presentan con los mas importantes en la mayoría de las aplicaciones móviles.

Nos apoyaremos de la equivalencia de*widgets* para poder determinar que *widgets* de navegación equivalentes consumen menos energía. Esto nos garantiza que en el diseño de aplicaciones los *widgets* de navegación mantendrán funcionalidad, usabilidad y además van tener un mejor desempeño energético.

CAPÍTULO 4

Medición de energía para plataformas móviles

En este capítulo se detalla los métodos aplicados en la literatura para medir el consumo de energía en un dispositivo móvil. Dado que el consumo energético puede variar dependiendo de las características de las aplicaciones, en la Sección 4.1 se explica cómo está compuesta la arquitectura de una aplicación móvil, se abordan las capas de presentación, de negocios y de datos. En la Sección 4.2 se describen cada tipo de aplicación: nativa, cliente e híbrida. Por último en la Sección 4.3 se explican los métodos para medir el consumo de energía, tanto a nivel *software* como a nivel de *hardware*, mostrando las ventajas y desventajas de cada una de ellas.

4.1. Arquitectura de una aplicación móvil

Una aplicación móvil está estructurada en tres capas que consisten en la capa de presentación, la capa de negocios y la capa de datos. En la Figura 4.1 se ilustra la arquitectura de una aplicación móvil con los componentes agrupados a cada una de las capas.

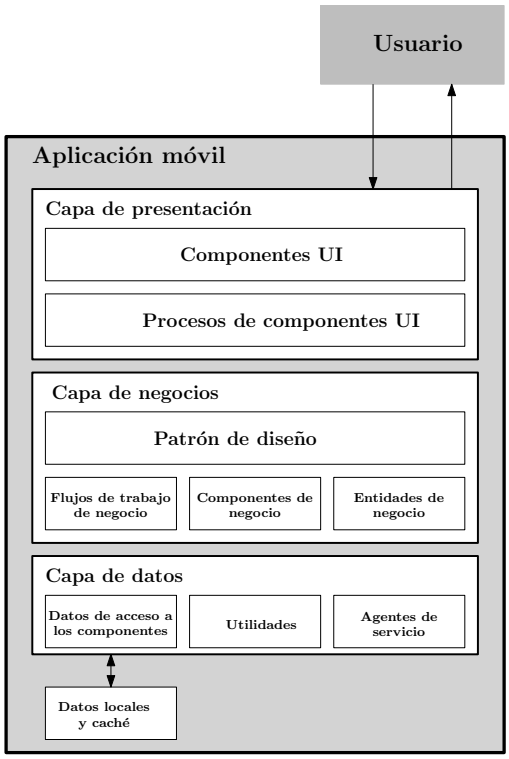


Figura 4.1. Arquitectura de una aplicación móvil [42].

4.1.1. Capa de presentación

Contiene los componentes gráficos de la GUI y gestionan la interacción con el usuario. Esta capa incluye los controles de entrada para el usuario y la pantalla, además de los componentes que organizan la interacción del usuario [29]. Esta capa está compuesta por dos componentes: la GUI y la de los procesos de los componentes de la GUI. La GUI proporciona una forma para que los usuarios interaccionen con la aplicación, este componente procesa y da formato a los datos que muestra. Los procesos de componentes de la GUI que sincronizan y organizan las interacciones del usuario, este componente es muy útil cuando se tiene una interfaz de usuario complicada.

4.1.2. Capa de negocios

Esta capa establece las reglas de la aplicación que va a seguir para utilizar la información, es decir, predecir y definir el flujo de trabajo, así como los objetos que serán relevantes para la aplicación [29]. Esta capa está compuesta por cuatro componentes: la aplicación *facade*, las entidades de negocio y el flujo de trabajo de negocio. La aplicación *facade* combina múltiples operaciones de negocios en una sola operación basada en mensajes. La de negocios procesa las reglas de negocio e interaccionan con los componentes de acceso a datos. Las entidades de negocios se utilizan para pasar los datos entre los demás componentes y el flujo de trabajo de negocios que definen y coordinan el funcionamiento de los procesos de negocios en varios pasos y pueden implementarse con herramientas de gestión de procesos de negocios.

4.1.3. Capa de datos

En esta capa se establece qué tipos de datos son necesarios para la aplicación, además proporciona el acceso a estos en el almacenamiento de algún tipo que puede ser una base de datos, una referencia a un objeto, entre otros [29]. Esta capa está compuesta por tres componentes: los lógicos de acceso a datos, las utilerías y agentes de servicio. Los lógicos de acceso a datos son los componentes de acceso a datos lógicos abstractos y hacen que la aplicación sea más fácil de configurar y mantener. Las utilerías ayudan en la manipulación, transformación y acceso a los datos dentro de la capa. Los agentes de servicios aíslan la aplicación de llamar a diversos servicios y puede proporcionar servicios adicionales como el mapeo básico entre el formato de los datos expuestos por el servicio y el formato requerida por la aplicación.

4.2. Tipos de aplicaciones móviles

Las diferentes plataformas móviles pueden ejecutar las aplicaciones de diversas formas, dependiendo en la plataforma que se encuentre. Por ejemplo, *Android* y *Windows Mobile* son plataformas que están basadas en máquinas virtuales (JVM para *Andorid* y CLR para *Windows Mobile*), IOS se ejecuta sobre una plataforma nativa y WebOS es una plataforma basada en tecnología web que utiliza las tecnología HTML5 JavaScript y CSS. Por tanto, podemos ver diferentes tipos de aplicaciones que dependen en donde se ejecute la capa de negocios y la capa de datos. Estas capas se pueden ejecutar desde un cliente, un servidor o de manera híbrida donde una parte de las capas pueden ejecutarse en el cliente y la otra parte en un servidor.

4.2.1. Nativa

La aplicación nativa es la más común. También puede llamarse aplicación de plataforma ya que tiene que ser desarrollada y compilada en cada plataforma. Esta aplicación cuenta con las tres capas como se observa en la Figura 4.2.

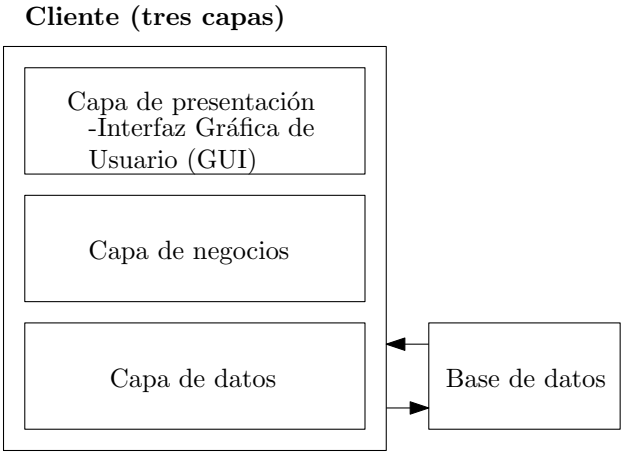


Figura 4.2. Arquitectura de una aplicación nativa [24].

Una aplicación nativa esta construida específicamente para un dispositivo y un sistema operativo, es decir para una plataforma móvil. Esta aplicación puede ser desarrollada en diversos tipos de lenguaje como Java, Swift y C++, entre otros. La aplicación nativa intenta aprovechar las características de la plataforma donde se ejecuta. Por ejemplo, el uso del acelerómetro, la cámara, el calendario, etc.

Las ventajas de este tipo de aplicación es ofrecer una mejor experiencia al usuario, un mejor diseño y aprovecha las características del dispositivo, tanto a nivel de *hardware* (la cámara y el GPS, etc.) como a nivel de *software* (calendario y contactos, entre otros). Además es una aplicación que siempre se encuentra disponible, ya que requiere una mínima conexión a Internet. Por último, es más simple de desarrollar debido a que solo se diseña para una plataforma. Sin embargo, una de las desventajas que limita a este tipo de aplicación, es que tiene poca portabilidad, incluso en diferentes versiones del S.O., el mantenimiento es muy costoso, puede tener procesos que no se puedan ejecutar en la plataforma, puede presentar problemas con la memoria al ejecutarse.

4.2.2. Cliente

Este tipo de aplicación es un modelo en el que las tareas se reparten entre el cliente y el servidor. El cliente es quien inicia las solicitudes, tiene una comunicación activa, espera y recibe respuestas de algún servidor e interacciona con el usuario median la GUI. El servidor espera las solicitudes del cliente, procesa la solicitud para enviar una respuesta y por lo general, acepta un gran número de clientes. Por tanto, el cliente envía un mensaje solicitando una petición a un servidor y éste envía uno o varios mensajes de respuesta. Este tipo de aplicación son conocidas generalmente como aplicaciones web. En este modelo de aplicación se distinguen dos tipos de aplicaciones: *thin-Client* y *fat-Client*.

Thin-Client

Este tipo de cliente no tienen códigos de la aplicación personalizada y confían completamente en el servidor para su funcionalidad, es decir, tanto el procesamiento como la obtención de datos depende del servidor y en el dispositivo sólo se visualizan los resultados requeridos por el usuario, por lo cual siempre es requerida una conexión a Internet. Por lo tanto, este tipo de cliente no depende en gran medida del sistema operativo del dispositivo móvil. La arquitectura de esta aplicación no tiene las capa de negocios y la capa de datos [24], sólo se tiene la capa de presentación como se observa en la Figura 4.3.

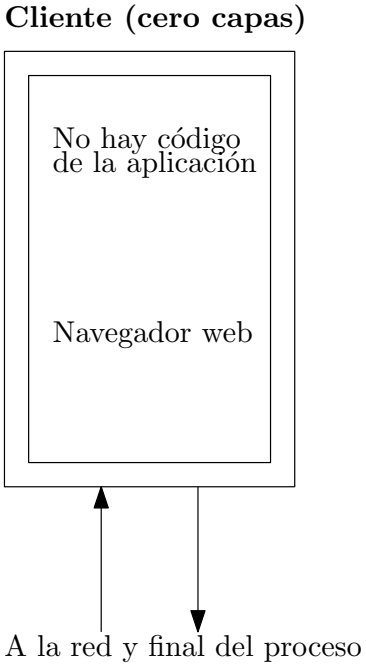


Figura 4.3. Arquitectura de una aplicación thin-Client [24].

Este tipo de cliente es más fácil de mantener y puede ejecutarse casi en cualquier tipo de dispositivo, por lo cual no hay necesidad de considerar la distribución de procesos entre el cliente y el servidor. Por otra parte, siempre

debe estar en comunicación con el servidor, el dispositivo debe encontrarse conectado a Internet para que la aplicación pueda actualizarse y obtener los datos requeridos.

Fat-client

Al igual que la *thin-Client*, este cliente necesita de un servidor, sin embargo puede funcionar independientemente por algún tiempo. Esto se debe a que sólo se conecta al servidor para descargar o actualizar datos, todo el procesamiento y funcionamiento depende del dispositivo móvil. La arquitectura de esta aplicación puede ser de una a tres capas, no obstante si se desea reutilizar código con mayor facilidad debe contar con más de dos capas. En la Figura 4.4 se muestra la arquitectura de este cliente [24].

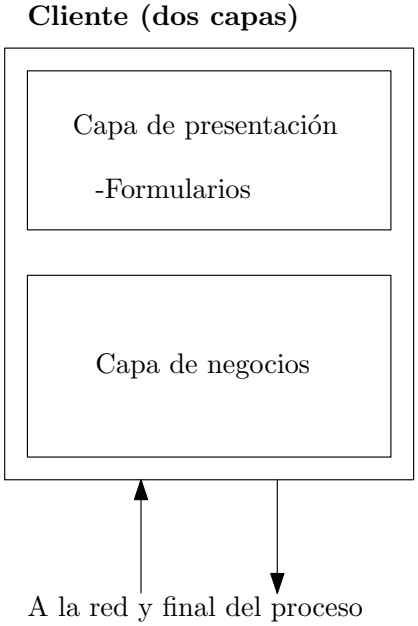


Figura 4.4. Arquitectura de una aplicación fat-Client [24].

La ventaja de utilizar este cliente es que le permite al usuario seguir trabajando si no se tiene una conexión a Internet. Por otro lado, dependen

en gran medida del tipo de dispositivo móvil y el sistema operativo por lo cual, es difícil mantener y soportar la aplicación en múltiples versiones. Además esta aplicación no aprovecha los recursos de la plataforma móvil en la que se ejecuta.

4.2.3. Híbrida (*Rich-Client*)

Una aplicación híbrida o *rich-Client* es una combinación de las aplicaciones nativas y cliente. Esta aplicación no necesitan ser instaladas o compiladas en alguna plataforma en específico. Se ejecuta sobre cualquier navegador móvil, por lo cual su diseño requiere que sea desarrollada principalmente de HTML5, CSS y JavaScript. Esta aplicación se asemeja a la nativa, ya que es capaz de aprovechar las características del dispositivo móvil en el que se esté ejecutando y, a diferencia de las aplicaciones cliente, pueden ejecutarse sin necesidad de encontrarse conectado a Internet. Como se observa en la Figura 4.5 la arquitectura de esta aplicación cuenta con tres capas. Sin embargo, tanto la capa de negocios como la capa de datos puede ejecutarse completamente o bien dividirse el procesamiento en el dispositivo o algún servidor.

La ventaja de utilizar este tipo de aplicación es la compatibildad que existe entre las diferentes plataformas móviles, lo que permite llegar a mayor usuarios con un menor esfuerzo. Por otra parte, los navegadores web son bastante estandarizados, lo que hace más fácil crearla, esto permite que sea más fácil de mantener y son menos costosas. Sin embargo, la capacidad gráfica de esta aplicación son limitadas ya que se visualizan como cliente y se requiere que el navegador sea capaz de ejecutar esta aplicación (no todos los navegadores soportan HTML5, CSS y JavaScript).

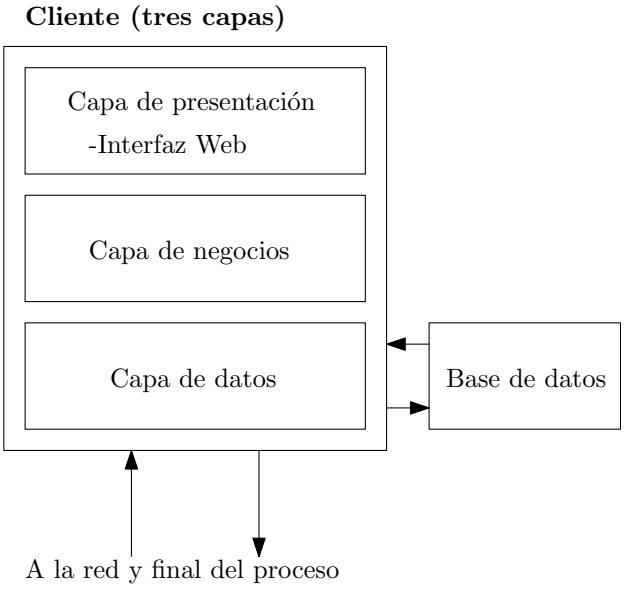


Figura 4.5. Arquitectura de una aplicación rich-Client [24].

4.3. Medición de energía en dispositivos móviles

Actualmente, el rendimiento de un sistema no sólo se denota por la capacidad de procesamiento de sus tareas, sino que también la eficiencia en el consumo de energía [40]. La potencia es la energía consumida durante una unidad de tiempo, la unidad para medir la potencia es el watt (W). La energía es el costo que puede ser medida por la potencia durante el tiempo de ejecución de una tarea, la unidad para medir la energía es el joule (J).

El uso de los dispositivos móviles ha aumentado a lo largo de los últimos años. Este aumento se debe a que realizan cada vez más diversas tareas como enviar y recibir correos, tomar fotografías y videos, procesar documentos y conectarnos a las redes sociales, entre otros. Por tanto, los dispositivos móviles cuentan con más componentes como el GPS, la cámara, un procesador multinúcleos y la interfaz de red con diferentes protocolos de

comunicación, etc. El uso de estos componentes incide directamente en la carga de la batería, es por esto que existe la necesidad de predecir el tiempo de carga de la batería y calcular el consumo de energía de una aplicación. Para medir el consumo de energía existen herramientas tanto de *software* como de *hardware* que nos permiten conocer la energía que esta utilizando un componente o una aplicación.

En la Subsección 4.3.1 se describe el mecanismo que ofrece la plataforma móvil para predecir el consumo de energía y potencia en sus diferentes componentes y que herramientas se han implementado para realizar esta tarea. En la Subsección 4.3.2 se especifican algunas herramientas que se utilizan para medir directamente del dispositivo móvil la intensidad y estimar el consumo de la energía y la potencia. Por último, en la Subsección 4.3.3 se propone una metodología para llevar a cabo la medición de energía y potencia para determinar cual es el consumo total de uno o varios componentes del dispositivo móvil al utilizar una aplicación.

4.3.1. Medición de energía basada en *software*

En esta sección se aborda como se administra la energía de un dispositivo móvil desde el *kernel* del sistema operativo y como se obtiene información a través de la interfaz avanzada de configuración y energía. También se presenta algunas herramientas que a través de diferentes métodos estiman la energía que se esta utilizando.

Administración de energía en un dispositivo móvil

Para entender como obtener la información de consumo de energía en un dispositivo móvil, es necesario entender cómo funcionan las políticas de gestión de energía mediante el control de estados de los componentes del dispositivo. Desde el punto de vista de la arquitectura, el sistema es un conjunto de componentes que interaccionan dentro del sistema [20]. La actividad de los componentes está coordinado por el *kernel* del sistema

operativo que se comunica a través de los controladores del sistema. Por lo tanto el sistema operativo proporciona una Interfaz Avanzada de Configuración y Energía (ACPI, por sus siglas en inglés) que es el principal responsable de la gestión de los estados de funcionamiento [20]. Como se muestra en la Figura 4.6, la capa ACPI se encuentra entre el *kernel* y *hardware*.

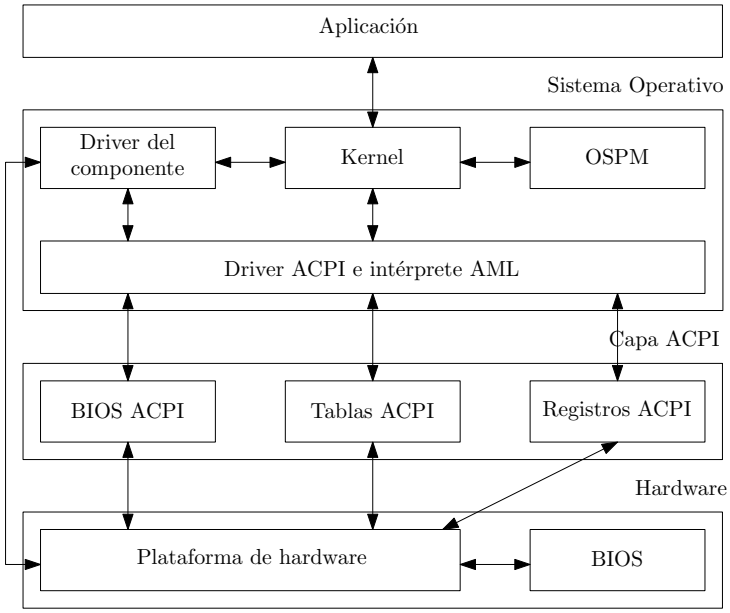


Figura 4.6. Configuración Avanzada e interfaz de energía (ACPI).

El *kernel* de Linux proporciona una interfaz que permite visualizar los registros para gestionar, analizar, dar seguimiento y depurar una aplicación. Esta interfaz es llamada *"Traceview"* [44].

Medición de energía con la capa ACPI

Estimar el consumo de energía mediante *software* en tiempo real no es una tarea trivial considerando que no se conoce la corriente que circula en la batería e incluso si la conociéramos, no sería suficiente para estimar la energía

de cada componente [20]. Bircher et al. [3] propone una técnica con minería de datos para estimar el consumo de potencia y de energía de los componentes desde el consumo total de energía del sistema. Por otra parte Kreiman et al. [20] propone un análisis de regresión lineal teniendo como variable independiente a los componentes del *hardware* y como variable dependiente a la corriente que circula en la batería. De la misma manera, Dolezal et al. [10] proponen un modelo matemático para predecir el consumo de energía sin la necesidad de realizar mediciones en tiempo real.

Se han desarrollado diversas herramientas para medir el consumo de energía y de potencia. En su mayoría se han implementado en plataformas con el S.O de *Andorid*. *Power Tutor* [51] estima el consumo de energía en tiempo real utilizando un modelo de regresión, para los componentes CPU, al pantalla LCD, el GPS, la antena y las interfaces de red, además es capaz de estimar la energía que consume una aplicación en cada componente. Sin embargo, *Power Tutor* no es capaz de medir la energía que consume cada proceso de la aplicación. *Eprof* [32] se ha enfocado en realizar un modelado de energía de granularidad fina, es decir que es capaz de medir el consumo de energía a nivel de proceso, permitiendo al desarrollador optimizar su aplicación. *AppScope* [49] es capaz de estimar el consumo de energía tanto de los componentes como de la aplicación a nivel de proceso.

4.3.2. Medición de energía basada en *hardware*

Medir la energía que consume una aplicación o un componente de la plataforma móvil no es fácil. Los circuitos que conectan los componentes con la alimentación de la batería son muy pequeños y manejan diferentes intensidades y voltajes. Por tanto, medir directamente la energía que esta consumiendo un componente de manera externa se vuelve una tarea complicada.

La energía eléctrica es la forma de energía que resulta de la existencia de un voltaje entre dos puntos permitiendo establecer una corriente eléctrica

entre ambos puntos durante un tiempo establecido. Se muestra la ecuación de la energía en 4.1, donde E_J representa la energía en joules, V es el voltaje, I es la intensidad (o corriente) y T es el tiempo total que duró la tarea, generalmente el tiempo se mide en segundos,

$$E_J = V\, I\, T. \tag{4.1}$$

Por lo tanto, la potencia es la energía consumida de una tarea durante una unidad de tiempo. En la Ecuación 4.2 se muestra la ecuación para calcular la potencia donde P_W es la potencia en watts.

$$P_W = \frac{E_J}{T}. \tag{4.2}$$

Instrumentos para medir la energía en dispositivos móviles

Para medir la energía desde *hardware* es necesario hacerlo desde la fuente de alimentación, es decir de la batería. Para realizar esta medición existen diferentes herramientas. Éstas pueden ser el osciloscopio, el multímetro y el *Power monitor*, por mencionar algunas.

Osciloscopio

El osciloscopio es un instrumento de visualización electrónico para la representación gráfica de señales eléctricas que pueden variar en el tiempo. Presenta los valores de las señales eléctricas en forma de coordenadas. El eje X representa el tiempo y el eje Y representa la tensión. El osciloscopio puede ser analógico o digital. En [4, 41] utilizan un osciloscopio para medir la energía. En la Figura 4.7 se muestra un circuito para medir la energía en un dispositivo móvil utilizando osciloscopio.

Para obtener la energía, es necesario aplicar la ley de Ohm que establece que el voltaje que aparece entre los extremos de un conductor determinado es proporcional a la intensidad de la corriente, es decir $I = \frac{V}{R}$. En la Figura 4.7

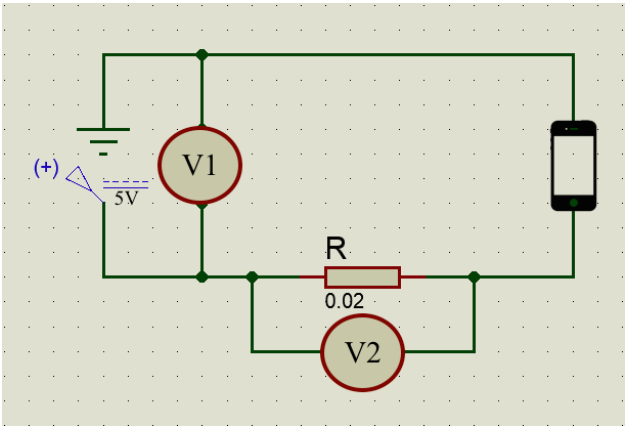


Figura 4.7. Circuito para medir energía con osciloscopio en un dispositivo móvil.

tenemos que $V1$ es el voltaje medido que esta siendo suministrado al circuito, $V2$ es el voltaje que pasa a través de la resistencia R. Por lo tanto, la Ecuación 4.1 se puede escribir como aparece en 4.3,

$$E_J = V1 \frac{V2}{R} T. \tag{4.3}$$

Multímetro

El multímetro es un instrumento eléctrico portátil para medir directamente magnitudes eléctricas activas (corrientes, voltajes y resistencia, entre otros). Estas medidas pueden realizarse para corriente alterna o corriente continua. Este instrumento puede ser análogo o digital. En [37, 8, 10] utilizan el multímetro para obtener la intensidad y el voltaje. En la Figura 4.7 se muestra un circuito para medir la energía en un dispositivo móvil utilizando el multímetro.

Para obtener la energía consumida durante un periodo, sólo basta con tomar la lectura de la intensidad y aplicar la Ecuación 4.1.

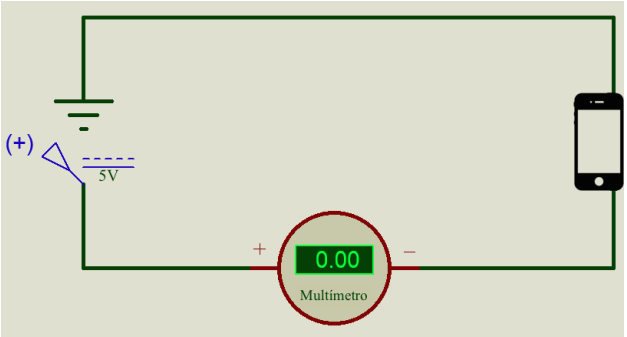
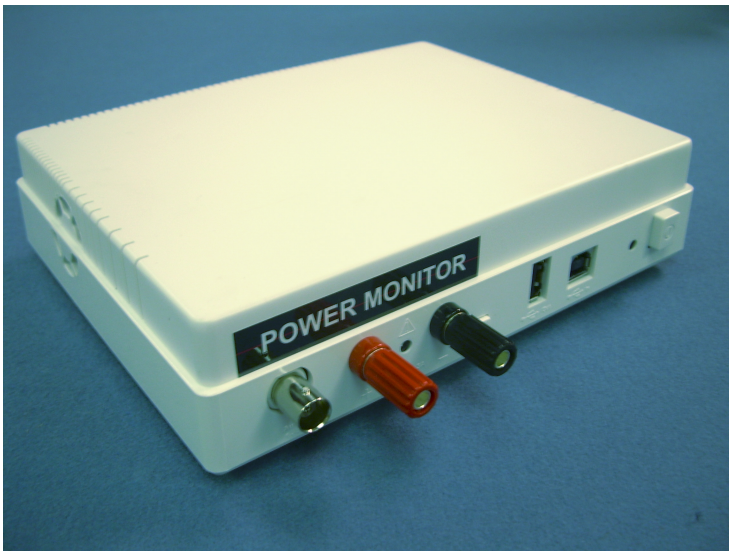


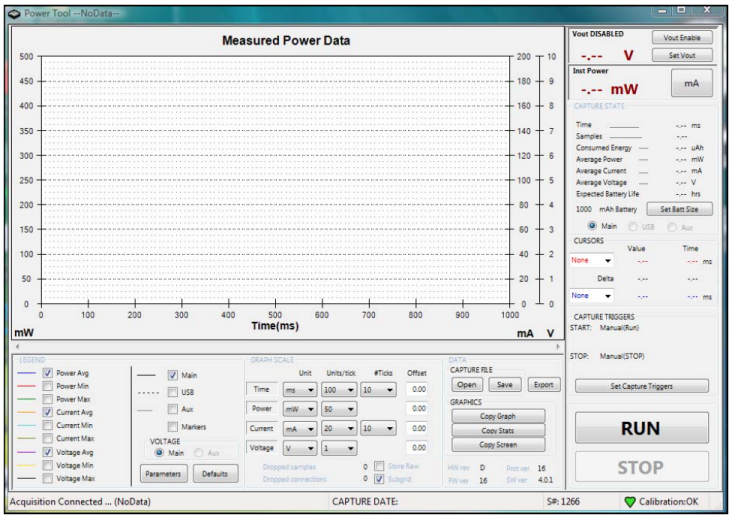
Figura 4.8. Circuito para medir corriente eléctrica con multímero en un dispositivo móvil.

Power monitor

Power monitor es un instrumento que, además de medir voltaje e intensidad, da una estimación en horas de la duración de la carga de batería y proporciona un *software* que permite mostrar los valores en tiempo real, esta herramienta sólo funciona para dispositivos con batería de litio de 4.5V y de hasta 3.0A. En la Figura 4.9 se muestra el instrumento *Power monitor* y la interfaz gráfica proporcionada por el desarrollador. En [31] utilizan este instrumento para medir la energía y además tener una aproximación de cuanto puede durar la carga de la batería. En la Figura 4.10 se muestra una implementación para medir la energía en un dispositivo móvil utilizando el *Power meter*.



(a) Instrumento *Power Monitor*



(b) Interfaz gráfica de la aplicación proporcionada por el desarrollador

Figura 4.9. Instrumento Power monitor en un dispositivo móvil[1].

[1]El lector interesado puede consultar la página del fabricante https://www.msoon.com/LabEquipment/PowerMonitor/ que explica detalladamente el uso del instrumento

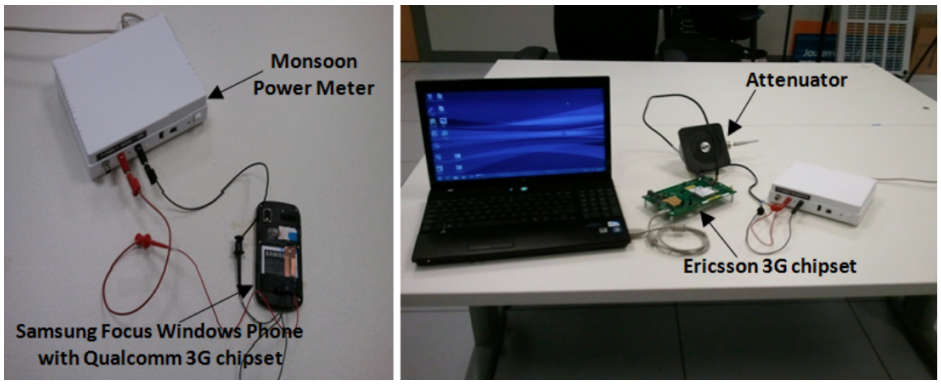


Figura 4.10. Circuito para medir energía con Power monitor en un dispositivo móvil [31].

4.3.3. Metodología para medir la energía en componentes de dispositivos móviles

Medir la energía de un componente de un dispositivo móvil no es fácil. Para medir el consumo de energía que utiliza uno o más componentes se propone seguir los pasos del método que se muestra en Figura 4.11. Esta propuesta mantiene coherencia con la metodología que se presenta en [1], ya que sólo se ajusta para el proceso particular que nosotros requerimos medir.

Primero se debe identificar los componentes del dispositivo que se requieren para ejecutar la aplicación. Después se inhabilitan todos los componentes activos, es decir aquellos componentes que el usuario puede controlar y ajustar, por ejemplo el *bluetooth* y el servicio de datos, con la finalidad de que no ocasione ruido durante el proceso para medir. Conectar el dispositivo móvil en el circuito. Medir la tensión que existe entre la batería y el dispositivo móvil. Ejecutar la aplicación para tomar la lectura de la corriente durante el tiempo de ejecución. Por último se deben realizar los cálculos para estimar la energía consumida.

Para estimar la energía que consumieron los componentes durante la ejecución es necesario tener una intensidad base I_B, es decir la intensidad en

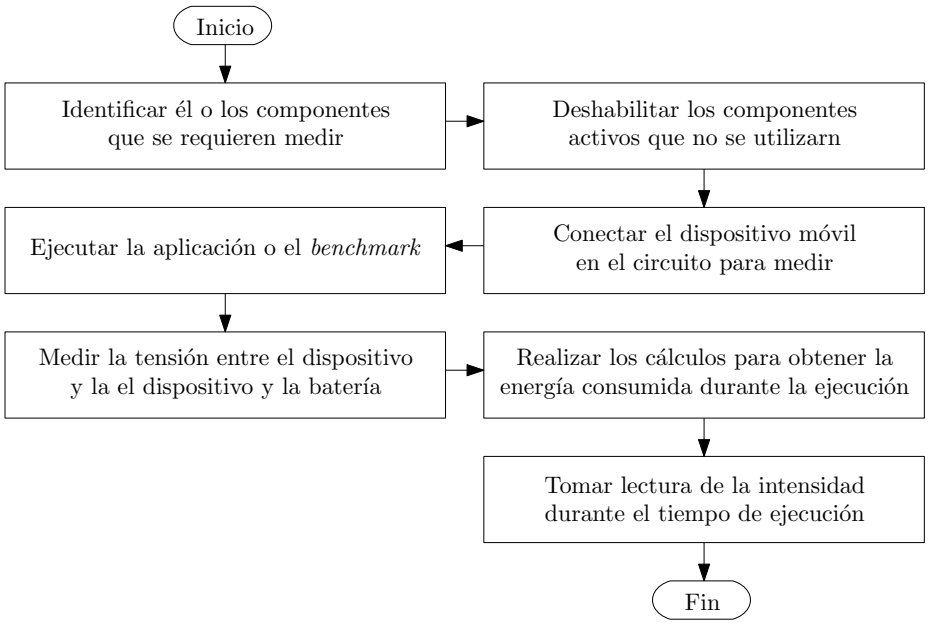


Figura 4.11. Método para medir la energía de los componentes del dispositivo móvil.

la que la plataforma móvil se encuentra en estado inactivo. En la Figura 4.12 esta intensidad esta señalada en las regiones I_b. Se toma un muestreo de m muestras para calcular la moda de la intensidad en el estado inactivo, es decir el valor que más se repite. Por lo tanto, para calcular la energía total de la aplicación en estado activo la denominamos E_T. Se considera el área bajo la curva de la gráfica que se encuentra sobre I_B. En la Ecuación 4.4 se muestra como calcular el área, donde n es el total de las muestras en estado activo de la aplicación, I es la intensidad, V es el voltaje suministrado por la batería (por lo general es constante) y d_t es el tamaño de paso representado en segundos,

$$E_T = \int_{i=1}^{n} (I_i - I_B) V d_t. \tag{4.4}$$

La Ecuación 4.4 la podemos aproximar. Primero se discretiza la gráfica en triángulos y rectángulos para cada muestra de manera uniforme en el tiempo, por lo tanto d_t siempre va a tener la misma magnitud. En la Figura 4.13 se observa un ejemplo del espacio discretizado. Por tanto, para cada muestra debemos calcular la energía E_i. Esto equivale a calcular el área del espacio discretizado. Esto lo obtenemos sumando el área del triángulo y el rectángulo multiplicado por el V y d_t, ver Ecuación 4.5. Entonces, la E_T aproximada es la suma de energía de todas las muestras E_i, como se observa en la Ecuación 4.6,

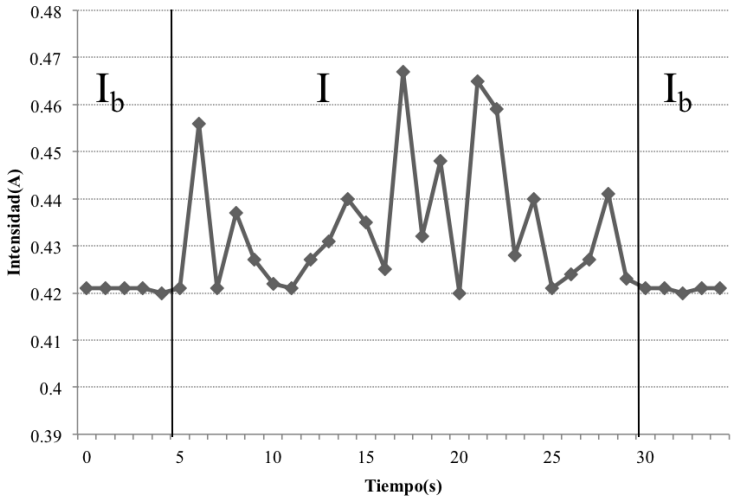


Figura 4.12. Ejemplo del comportamiento de la intensidad de la pantalla y el CPU ejecutando una aplicación con el widget de menú fijo.

$$E_i \cong \frac{|I_i - I_{i-1}|}{2} V d_t + (\min(I_{i-1}, I_i) - I_B) V d_t \tag{4.5}$$

$$E_T \cong \sum_{i=1}^{n} \frac{|I_i - I_{i-1}|}{2} V\, d_t + (\min(I_{i-1}, I_i) - I_B)\, V\, d_t. \tag{4.6}$$

Por último, para obtener la potencia sólo basta con dividir la energia E_T entre el tiempo T que duró la ejecución de la tarea, ver Ecuación 4.7,

$$P_T = \frac{E_T}{T}. \tag{4.7}$$

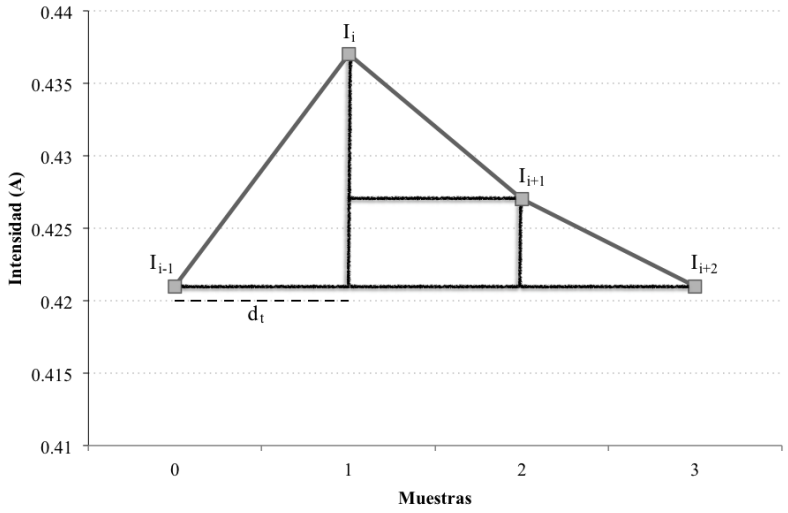


Figura 4.13. Ejemplo de una gráfica discretizada en el tiempo.

CAPÍTULO 5

Experimentos y resultados

En este capítulo se describe como se desarrollaron los experimentos para obtener estimar la energía que consumen algunos *widgets* de navegación y los resultados correspondientes. Se inicia, en la Sección 5.1 con la descripción de la infraestructura que se utilizó para llevar a cabo los experimentos y el método que se implementó para obtener la energía. En la Sección 5.2 se realiza una descripción de los *widgets* de navegación que se proponen y como la GUI cambia de estado al interaccionar con éstos. En la Sección 5.3 se explican los prototipos que se diseñaron identificando los *widgets* de navegación que se utilizaron, además se especifican las tareas de cada prototipo desarrollado y las tareas que se tomaron en cuenta para medir el consumo de energía en el dispositivo móvil. En la Sección 5.4 se muestran los resultados que se obtuvieron. Para finalizar con una breve discusión de los resultados en la Sección 5.5

5.1. Infraestructura y método para medir la energía

Para medir la energía de los componentes de un dispositivo móvil cuando se encuentra en estado activo se necesita una infraestructura que cuente con las características necesarias para que ejecute las aplicaciones y medir la energía.

5.1.1. Infraestructura en *hardware*

Para el estudio del consumo energético en *widgets* de navegación se utilizó la siguiente infraestructura para medir la energía.

- Raspberry Pi B+: ésta plataforma cuenta con un procesador Core ARM11 700 MHz, una memoria RAM de 512 MB.
- Pantalla táctil TFT 3.5" 320 × 480px
- Multímetro digital *True-RMS 46-Range*

Se eligió la plataforma Raspberry Pi con el sistema operativo Raspbian debido a que el *kernel* que maneja y las características en *hardware* son semejantes a diversos dispositivos móviles que hay en el mercado, ver Tabla 5.1 por mencionar algunos. Los resultados que se obtienen se pueden usar como base para estudios posteriores de la interfaz gráfica de usuario.

Dispositivo móvil	Procesador	RAM (MB)	Pantalla
HTC One V	Dual core Qualcom 1 GHZ	512	3.7" TFT
Samsung galaxy young 2	Broadcom 1 GHz	512	3.5" TFT
LG Joy	Quad core 1,2 GHz	512	4" IPS

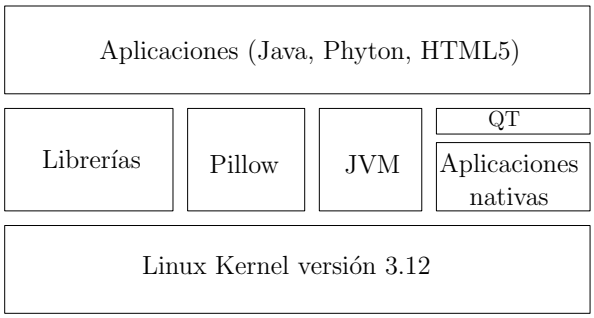
Tabla 5.1. Dispositivos móviles semejantes al hardware de la Raspberry pi B+.

5.1.2. Infraestructura en *software*

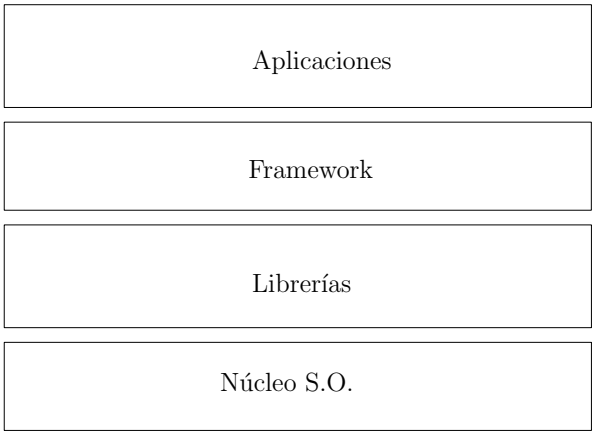
Para ejecutar los prototipos en las diferentes tipos de aplicaciones se requirió el siguiente *software*: S.O. "Rapsbian wheezy", los compiladores de "QT" para los prototipos de las aplicaciones nativas y de Java para los prototipos de las aplicaciones nativas con máquina virtual, las máquinas virtuales de "Java HotSpot" para ejecutar los prototipos que requieren máquina virtual y de "QEMU" para virtualizar el S.O "Chromium", los navegadores "Epiphany" para ejecutar las aplicaciones *rich-Client* de manera nativa y "Opera mini" para ejecutar las aplicaciones *rich-Client* con máquina virtualy los prototipos que manejen diferentes tipos de *widgets* de navegación (en la Sección 5.2 se detallan los *widgets* propuestos).

El S.O. Rapsbian wheezy permite ejecutar aplicaciones en entorno nativo y sobre una máquina virtual. En la Figura 5.1 se muestran las arquitecturas del S.O. Android, iOS y Rapsbian wheezy para compararlos. Se observa que la versión del *kernel* entre Android y Raspbian wheezy son similares.

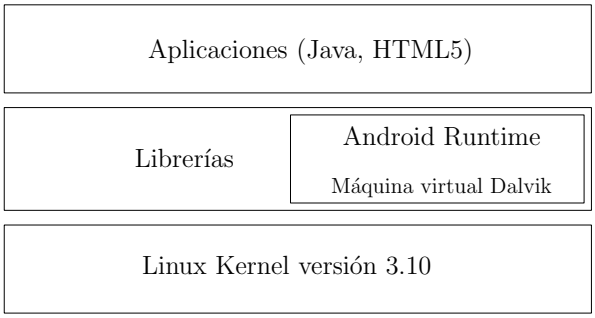
Cada S.O. ejecuta sus aplicaciones de manera particular. Las aplicaciones en el S.O. IOS no requieren una máquina virtual. Por lo tanto, los eventos y el despliegue de la GUI se maneja a través de los *frameworks* que proporciona el sistema. Las aplicaciones en el S.O. Android se ejecutan en una máquina virtual llamada *Dalvik*. Los eventos y el despliegue la GUI se describen en archivos XML que los instancia con una clase de java. Las aplicaciones en Raspbian wheezy ejecuta las aplicaciones nativas con máquina virtual (Java) y sin ella. Para las aplicaciones nativas sin máquina virtual los eventos y el despliegue de la GUI se manejan a través de bibliotecas que proporciona QT. Para las aplicaciones naticas con máquina virtual los eventos y el despliegue de la GUI se maneja con la clase *Jframe* que proporciona java. Por último, para las aplicaciones *rich-Client* los eventos y el despliegue de la GUI se realiza a través de un intérprete, es decir un navegador que sea capaz de ejecutar código HTML5 y JavaScript.



(a) Arquitectura general del S.O. Raspbian



(b) Arquitectura general del S.O. iOS



(c) Arquitectura general del S.O. Android

Figura 5.1. Arquitectura general del S.O. Raspbian, Androdid e IOS.

5.1.3. Método para medir el consumo de energía

Para obtener el comportamiento de energía de los *widgets* de navegación se utilizó la medición basada en *hardware*. Para ello, utilizamos el multímetro para obtener la intensidad en tiempo real que está utilizando el dispositivo móvil. Se utilizó el siguiente método.

1. Se implementó el circuito que se muestra en 4.8. Se utilizo una batería portátil que suministra 4.8 V. En la Figura 5.2 se muestra el circuito implementado.

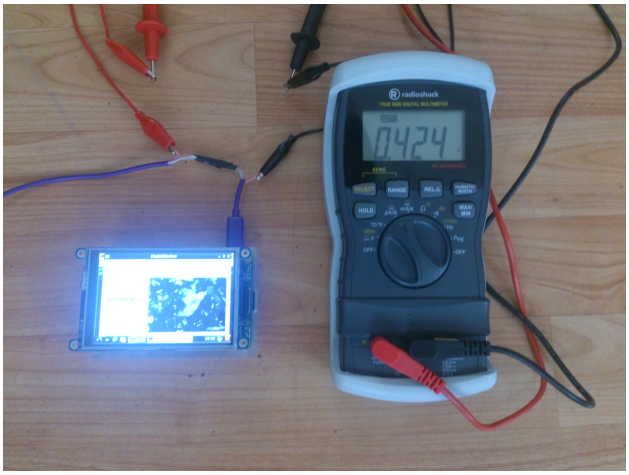


Figura 5.2. Implementación para medir el consumo de energía de un dispositivo móvil.

2. Para cada prototipo en cada tipo de aplicación se ejecutaron treinta pruebas. Dado que se requiere medir los *widgets* de navegación, la plataforma móvil se trabajo sólo con la pantalla al 100 % del nivel de brillo y el CPU.

3. Se calculó la energía consumida de cada cada prueba como se específica en la metodología descrita en la Subsección 4.3.3.

4. Se obtuvo un promedio de las treinta ejecuciones de cada prototipo para estimar la energía consumida de cada *widget* de navegación.

5.2. Descripción de los *widgets* de navegación propuestos

Los *widgets* de navegación que se consideraron con la relación de equivalencia son: menú fijo contra lista vertical, galería en miniaturas contra galería en *grid* y galería en carrusel contra lista de páginas [11]. En esta sección se describen cada uno de estos *widgets* y como la GUI cambia de estado al interaccionar con ellos.

5.2.1. Menú fijo

El menú fijo está compuesto por una lista que se despliega. Para seleccionar un elemento de la lista es necesario dar clic en el menú para desplegar la lista y elegir el elemento que se desea para que la GUI sea actualizada. En la Figura 5.3 se muestra un ejemplo del *widget* menú fijo.

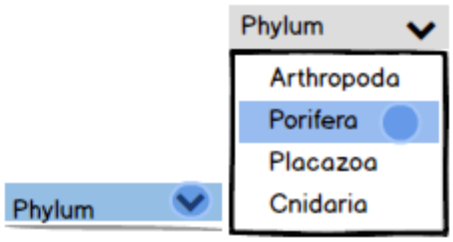


Figura 5.3. Ejemplo de menú fijo.

5.2.2. Lista vertical

La lista vertical esta compuesto por una serie de etiquetas de texto que son los elementos de la lista. Este tipo de *widget* permite que los elementos

sean visibles y, si la lista es grande, con la ayuda de un *scroll* permita ver los elementos que no se encuentran visibles. Por lo tanto, para que la GUI sea actualizada, sólo basta con seleccionar un elemento de la lista. En la Figura 5.4 se muestra un ejemplo del *widget* lista vertical.

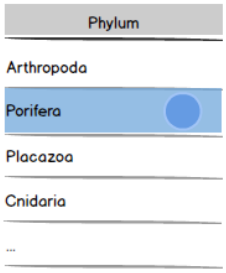


Figura 5.4. Ejemplo de lista en vertical.

5.2.3. Lista en miniaturas

La lista en miniaturas se presenta como una serie de etiquetas y cada una está asociada a una imagen y un texto. Al seleccionar una etiqueta de la lista la GUI cambia de estado. En la Figura 5.5 se muestra un ejemplo del *widget* de lista en miniaturas.

Figura 5.5. Ejemplo de lista en miniaturas.

5.2.4. Galería en *grid*

La galería en *grid* se muestra como una cuadrícula. Cada cuadro contiene una etiqueta que esta compuesta por una imagen y un texto. Al seleccionar una etiqueta de la galería la GUI cambia de estado. En la Figura 5.6 se muestra un ejemplo del *widget* de galería en grid.

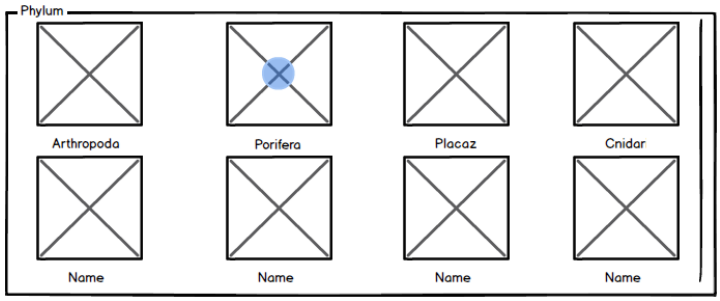


Figura 5.6. Ejemplo de galería en grid.

5.2.5. Galería en carrusel

La galería en carrusel está compuesta por una etiqueta y dos botones. La etiqueta la conforma una imagen y un texto. Los botones permiten navegar a través de la lista, ya sea hacia la izquierda o hacia la derecha las veces que el usuario lo requiera. Cada vez que se da clic en alguno de los botones o se selecciona una etiqueta la GUI cambia de estado. En la Figura 5.7 se muestra un ejemplo del *widget* de galería en carrusel.

5.2.6. Lista en páginas

La lista en páginas se presenta de manera similar a la lista vertical (ver Figura 5.4), sin embargo cada vez que se selecciona un elemento de la lista ésta se refresca para presentar otros elementos que pertenecen a otra lista, por lo tanto la GUI es actualizada. Este *widget* es ideal para dispositivos con pantallas pequeñas.

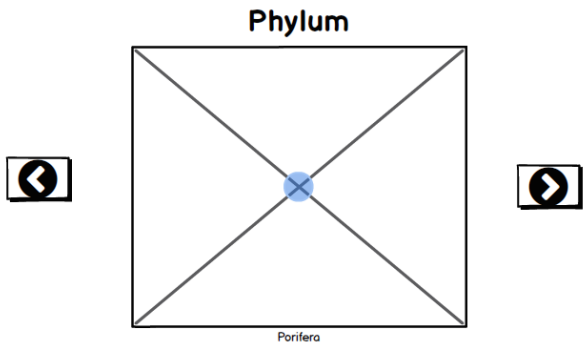


Figura 5.7. Ejemplo de galería en carrusel.

5.3. Prototipos y tareas

Como se menciona en el Capítulo 2, existen tres estados en que se puede encontrar un dispositivo móvil: activo, inactivo y suspendido. Sin embargo, cuando el dispositivo se encuentra en estado activo consume la mayor parte de energía, siendo la pantalla el componente que utiliza la mayor parte de energia ver Figura 2.4. Dado que la GUI esta asociado a la pantalla, al *framebuffer* y, dependiendo del dispositivo móvil, a la pantalla táctil. Por lo tanto, al desplegar o actualizar la GUI y que el usuario interáctue con el dispositivo a través de los diferentes gestos, ocasiona que consuma energía los componentes que están asociados a la GUI.

La interacción entre el usuario y el dispositivo móvil existe a través de la GUI que se despliega en la capa de presentación. Esta interacción se realiza a través de los diferentes *widgets*. Los *widgets* de navegación permiten al usuario desplazarse por la aplicación realizando diversas tareas. Estos *widgets* son importantes porque permiten cambiar el estado de la aplicación, es decir permiten navegar a través de las diferentes pantallas que componen la aplicación móvil. Por lo tanto, los *widgets* de navegación son de los más utilizados en todas las plataformas móviles.

Como se ha mencionado anteriormente, los *widgets* no son universales en todas las plataformas ni en las diferentes vistas, por lo que se debe realizar

una relación de equivalencia entre ellos. Por lo que el estudio de consumo de energía que se realiza en este trabajo se enfocará sólo en algunos *widgets* de navegación.

5.3.1. Prototipos

Los prototipos que se desarrollaron muestran la funcionalidad y la usabilidad de cada uno de los *widgets* de navegación. Los *widgets* de navegación que se consideraron con la relación de equivalencia para los prototipos son: menú fijo contra lista vertical, galería en miniaturas contra galería en *grid* y galería en carrusel contra lista de páginas [11].

Para el desarrollo de los prototipos se consideraron los siguientes tipos de aplicaciones: nativas sin máquina virtual, nativas con máquina virtual, *rich-Client* sin máquina virtual y *rich-Client* con máquina virtual.

Los prototipos se desarrollaron utilizando diferentes entornos de programación, esto con la finalidad que se ejecuten en los tipos de aplicaciones antes mencionadas. En la Tabla 5.2 se indican dichos entornos de programación y para que tipo de aplicación fue utilizado.

Aplicación	**Entorno de programación**
Nativas sin máquina virtual	C++ con GUI QT
Nativas con máquina virtual	JAVA
Rich-Client sin máquina virtual	HTML5, JavaScript y CSS3
Rich-Client con máquina virtual	HTML5, JavaScript y CSS3

Tabla 5.2. Entornos de programación para aplicaciones móviles.

Para la ejecución de la aplicación *Rich-Client* sin máquina virtual se utilizó el navegador Epiphany. Para la aplicación *Rich-Client* en un S.O. virtualizado se utilizó el navegador Chromium sobre la máquina virtual QEMU. Para la aplicación *Rich-Client* con máquina virtual se utilizó el navegador Opera con *Java Virtual Machine*.

Para realizar los prototipos se consideró un conjunto de pruebas base para navegar a través de una taxonomía de animales en seis niveles ordenados por el filo, la clase, el orden, la familia, el género y el tipo. Esta taxonomía es robusta y nos permite generar un *widget* por cada nivel. A pesar de que el diseño en varios niveles no entra en las recomendaciones de HCI[1], nos permite estresar los componentes que utiliza la GUI para poder estimar el consumo de energía. En la Figura 5.8 se muestra parte de la taxonomía que se utilizó.

5.3.2. Tareas

Una tarea es un conjunto de acciones. Una acción, para este trabajo, se considera que es un gesto (ver Figura 3.6). Para cada *widget*, de acuerdo a los estándares de HCI, para cada acción que realice el usuario debe haber una retroalimentación, por ejemplo si se da clic en un botón este debe de cambiar su color. Por lo tanto, la acción debe tener una repercusión en el *framebuffer*, provocando que cambie de estado la aplicación.

Por lo tanto, en este trabajo, consideramos que la tarea que debe realizar en cada prototipo es mostrar una imagen de un animal. Esto se logra navegando a través de los niveles de la taxonomía hasta llegar al nivel de "tipo".

En la Tabla 5.3 se resume la cantidad de acciones que se requieren realizar por cada *widget* así como la cantidad de acciones total para completar la tarea. En el *widget* de galería en carrusel n es la cantidad de clics que el usuario dé hacia la izquierdo y /o derecha.

[1]La recomendación de HCI es que la navegación debe ser máximo a dos niveles de profundidad.

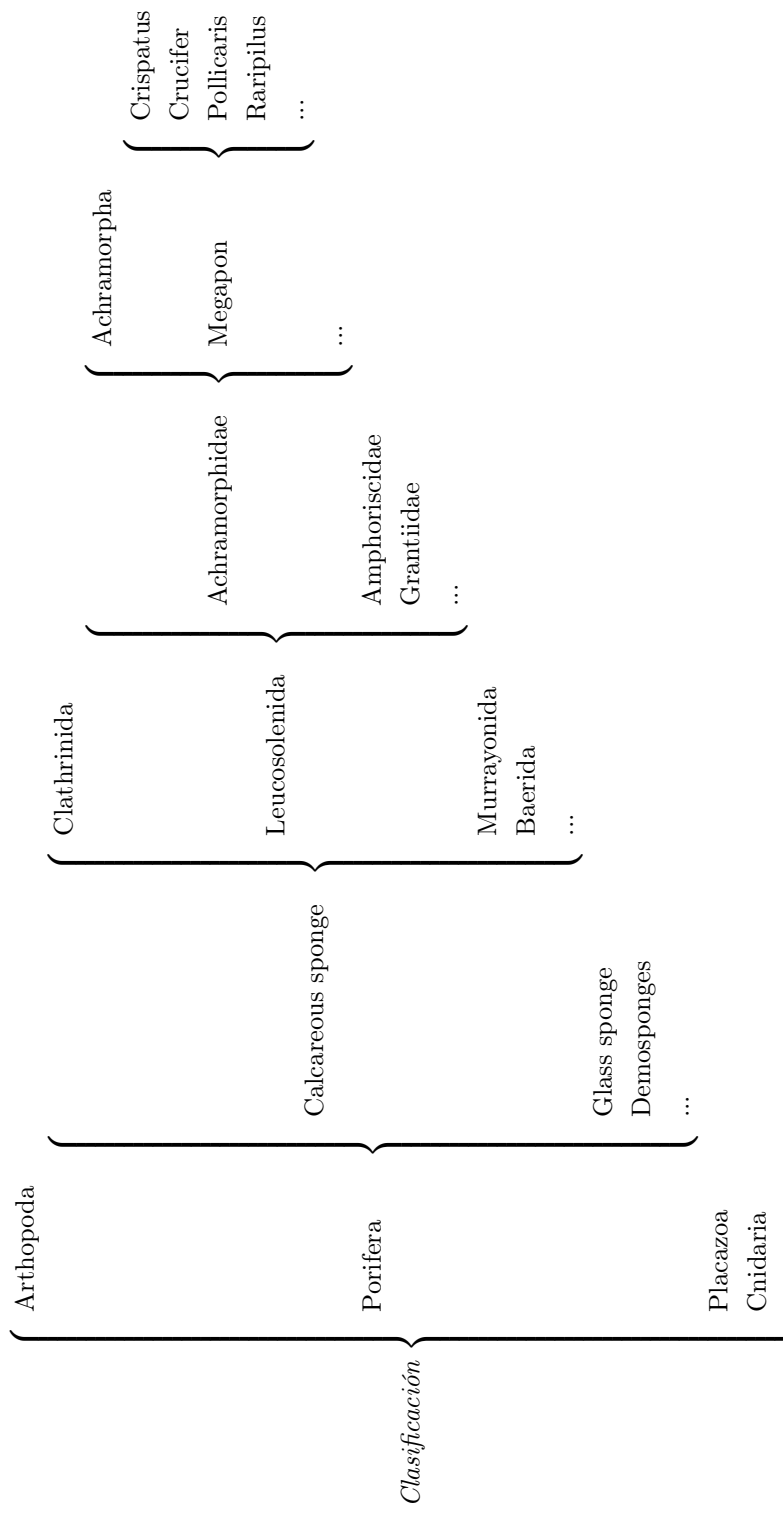


Figura 5.8. Taxonomía de animales.

Widget	Acciones por *widget*	Acciones por tarea
Menú fijo	2	12
Lista vertical	1	6
Lista en miniaturas	1	6
Galería en *grid*	1	6
Lista en páginas	1	6
Galería en carrusel	$n+1$	$(n+1) \times 6$

Tabla 5.3. Cantidad de acciones para cada prototipo.

A continuación se detalla la navegación de cada uno de los *widgets* de navegación seleccionados con la finalidad de mostrar las acciones que se deben ejecutar completar la tarea. Para cada *widget* propuesto se muestra el esqueleto de la GUI que se implementaron en las aplicaciones nativas y *rich-Client*. También el ejemplo de la tarea ilustrada para cada prototipo es el mismo, por lo que, en este caso, se debe llegar al tipo *"Crucifer"*. En la Figura 5.9 se muestra que se debe seleccionar de acuerdo al orden de la taxonomía anteriormente descrita.

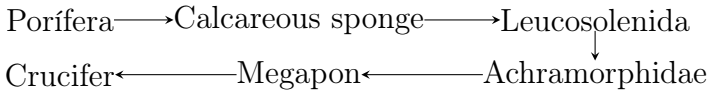


Figura 5.9. Ejemplo de selección para llegar al tipo "Crucifer".

Menú fijo

Para mostrar un animal en específico con el *widget* menú fijo se tiene que realizar las tareas que se enumeran a continuación.

1. Dar clic en la opción para desplegar el submenú.
2. Seleccionar un elemento del submenú.

Estás acciones se deben ejecutar para cada nivel de la taxonomía. En las Figuras 5.10 y 5.11 se muestran los esqueletos de las interfaces para los prototipos en aplicaciones nativas y *rich-client* respectivamente. La Figura 5.12 ilustra las acciones que se deben realizar para navegar a través de la taxonomía para completar la tarea.

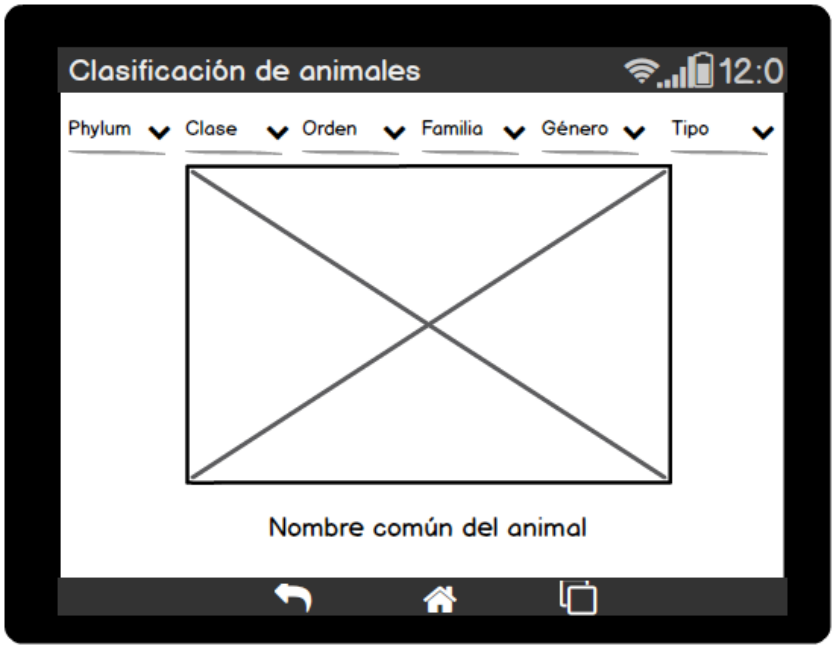


Figura 5.10. Esqueleto del la aplicación nativa con menú fijo.

(a) Esqueleto del la aplicación con el menú contraído

(b) Esqueleto del la aplicación con el menú expandido

Figura 5.11. Esqueleto del la aplicación rich-client con menú fijo.

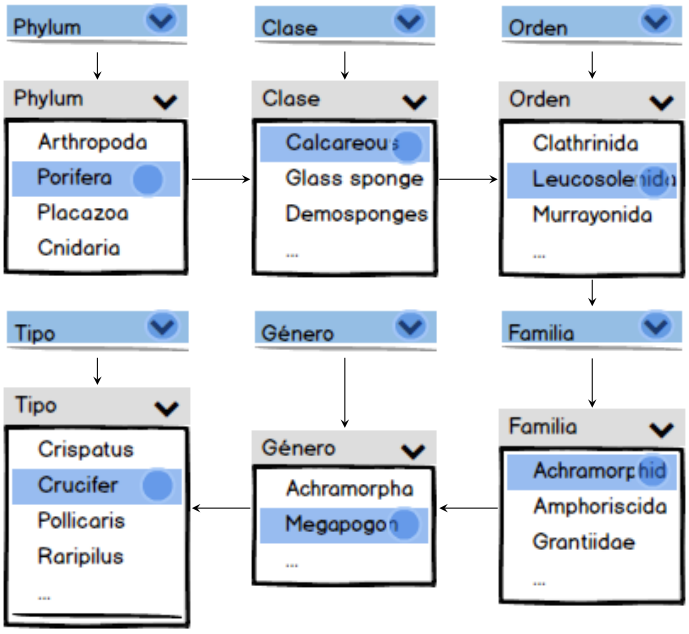


Figura 5.12. Acciones del widget de menú fijo.

Lista vertical

Para mostrar un animal en específico con el *widget* de lista vertical se tiene que realizar las siguientes acciones.

1. Dar clic en el elemento de la lista para desplegar los elementos de la siguiente de la siguiente lista de acuerdo al orden de la taxonomía.

Estas acciones deben se ejecutadas para cada nivel de la taxonomía. En las Figuras 5.13 y 5.14 se muestran los esqueletos de las interfaces para los prototipos en aplicaciones nativas y *rich-client* respectivamente. La Figura 5.15 ilustra las acciones que se deben realizar para navegar a través de la taxonomía para completar la tarea.

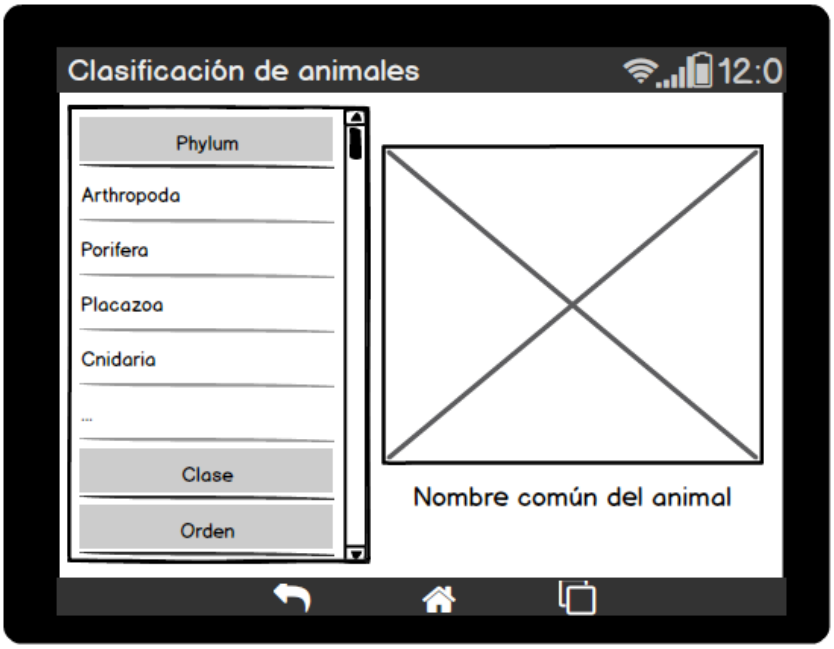


Figura 5.13. Esqueleto del la aplicación nativa con lista vertical.

Figura 5.14. Esqueleto del la aplicación rich-client con lista vertical.

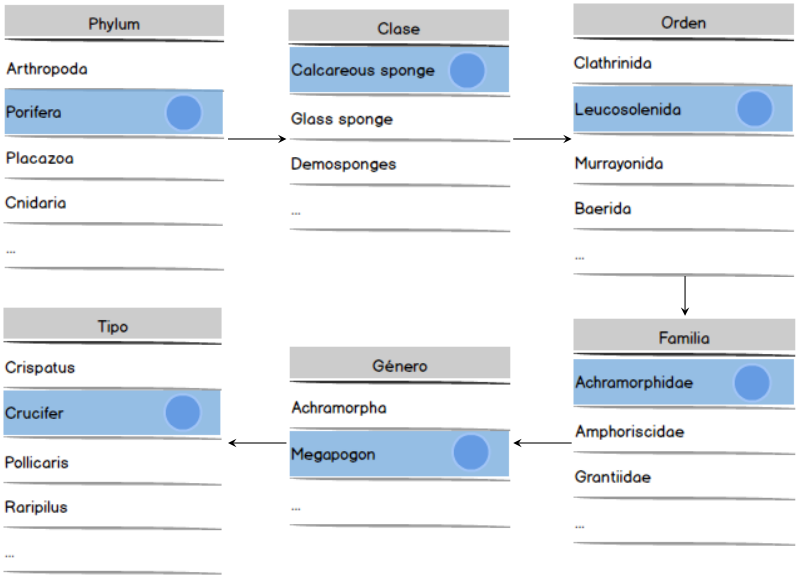


Figura 5.15. Acciones de lista vertical.

Lista de miniaturas

Para mostrar un animal en específico se necesita a través del *widget* lista en miniatura se tiene que realizar las siguientes acciones.

1. Dar clic en el elemento de la lista para desplegar los elementos de la siguiente lista de acuerdo al orden de la taxonomía.

Estas acciones deben se ejecutadas para cada nivel de la taxonomía. En las Figuras 5.16 y 5.17 se muestran los esqueletos de las interfaces para los prototipos en aplicaciones nativas y *rich-client* respectivamente. La Figura 5.18 ilustra las acciones que se deben realizar para navegar a través de la taxonomía para completar la tarea.

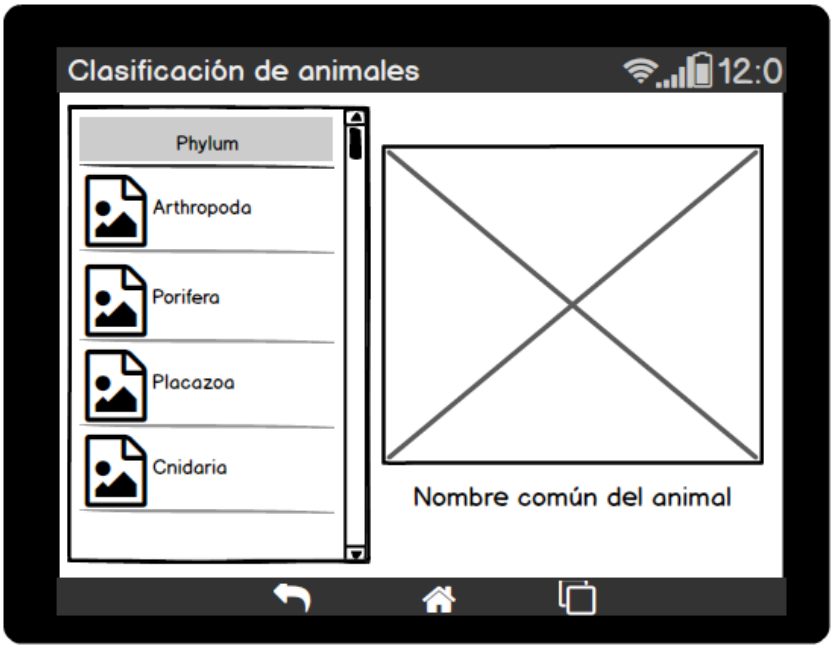


Figura 5.16. Esqueleto del la aplicación nativa con lista en miniatura.

Figura 5.17. Esqueleto del la aplicación rich-client con lista en miniatura.

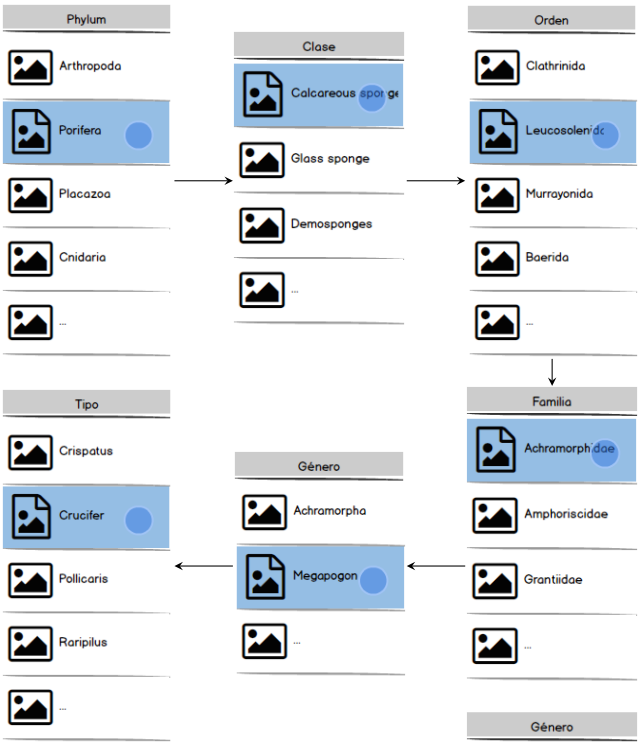


Figura 5.18. Acciones de lista de miniaturas.

Galería en *grid*

Para mostrar un animal en específico se necesita a través del *widget* galería en grid se tienen que realizar las siguientes acciones.

1. Dar clic en el elemento de la galería para desplegar los elementos de la siguiente galería de acuerdo al orden de la taxonomía.

Estas acciones deben se ejecutadas para cada nivel de la taxonomía. En las Figuras 5.19 y 5.20 se muestran los esqueletos de las interfaces para los prototipos en aplicaciones nativas y *rich-client* respectivamente. En la Figura 5.21 ilustra las acciones que se deben realizar para navegar a través de la taxonomía para completar la tarea.

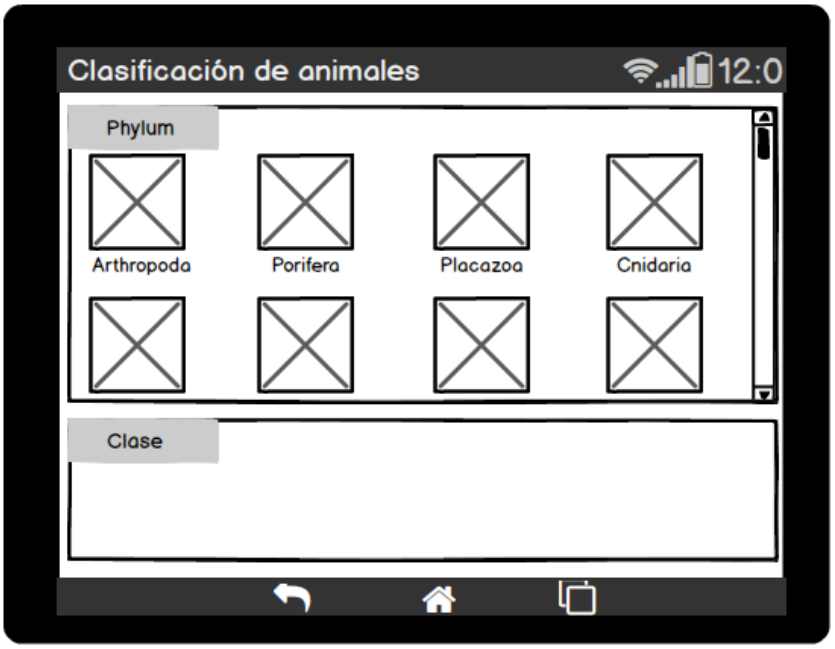


Figura 5.19. Esqueleto del la aplicación nativa con galería en grid.

Figura 5.20. Esqueleto del la aplicación rich-client con galería en grid.

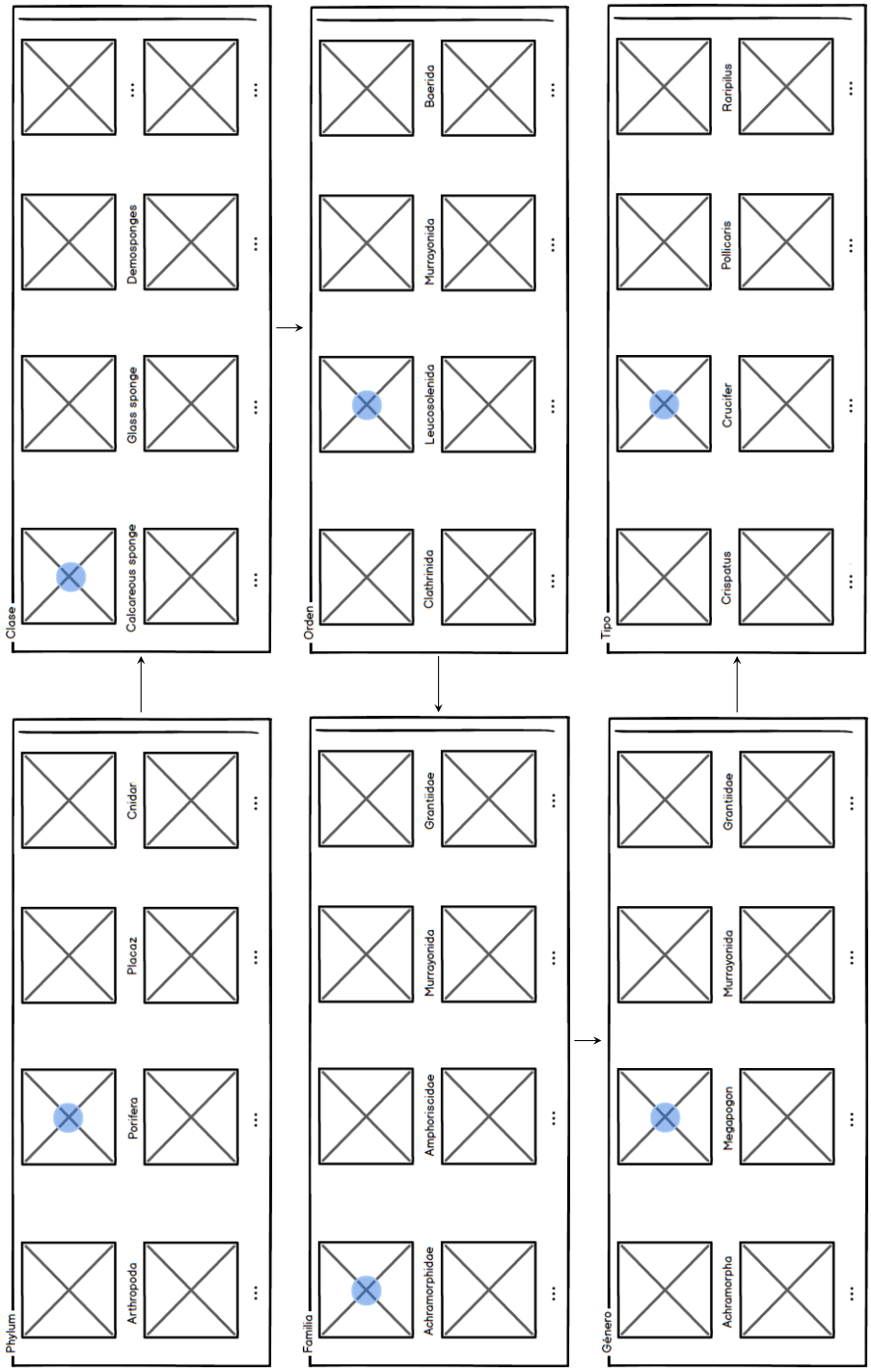


Figura 5.21. Acciones de galería en grid.

Galería en carrusel

Para mostrar un animal en específico se necesita a través del *widget* galería en carrusel se tienen que realizar las siguientes acciones.

1. Desplazarse hacia la derecha o izquierda hasta encontrar el elemento deseado.

2. Dar clic en el elemento de la galería para desplegar los elementos de la siguiente galería de acuerdo al orden de la taxonomía.

Estas acciones deben ser ejecutadas para cada nivel de la taxonomía. En las Figuras 5.22 y 5.23 se muestran los esqueletos de las interfaces para los prototipos en aplicaciones nativas y *rich-client* respectivamente. La Figura 5.24 ilustra las acciones que se deben realizar para navegar a través de la taxonomía para completar la tarea.

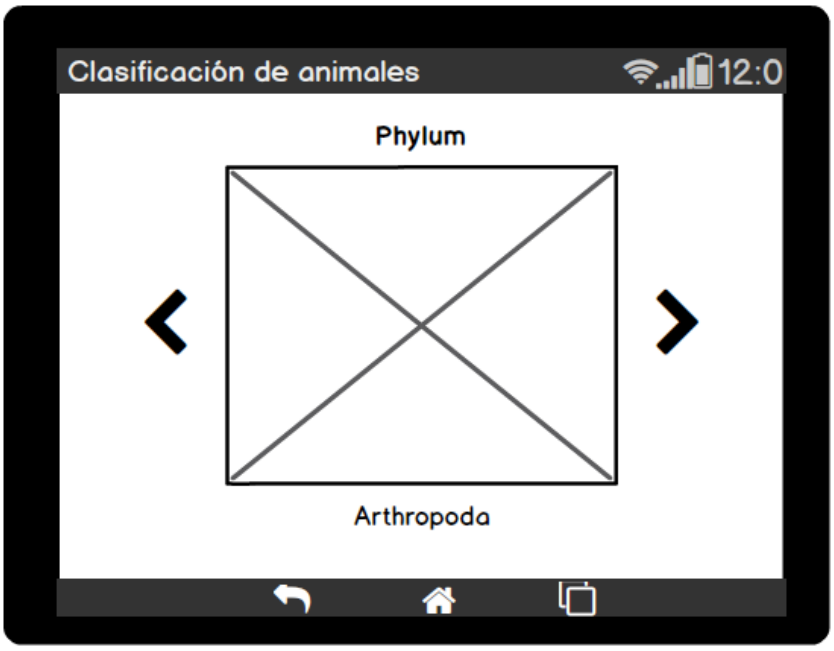


Figura 5.22. Esqueleto del la aplicación nativa con galería en carrusel.

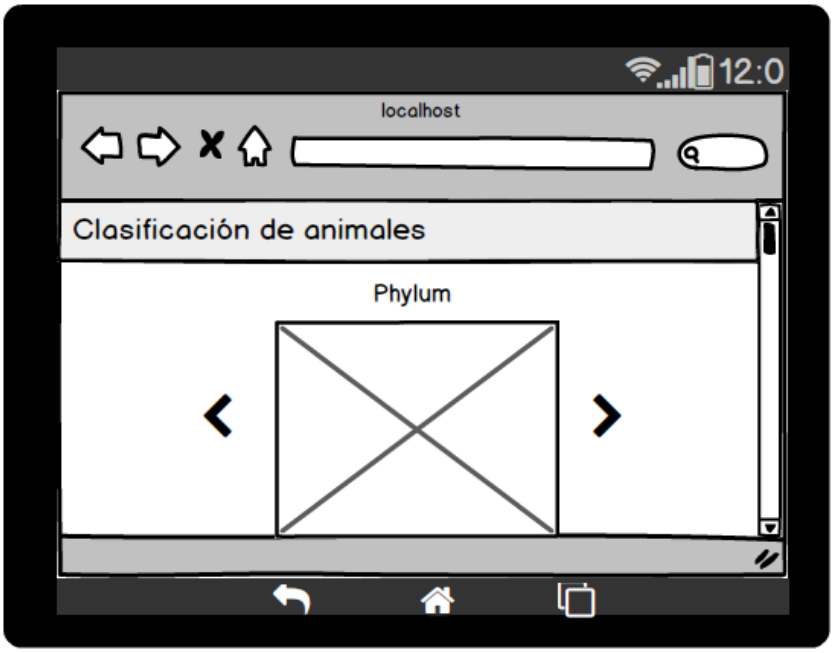


Figura 5.23. Esqueleto del la aplicación rich-client con galería en carrusel.

Lista en páginas

Para mostrar un animal en específico se necesita a través del *widget* lista en páginas se tienen que realizar las siguientes acciones.

1. Dar clic en el elemento de la lista desplegar la siguiente página con los elementos de la siguiente clasificación.

Estas acciones deben ser ejecutadas para cada nivel de la taxonomía. En la Figura 5.26 se muestra el esqueleto de la interfaz que se utilizó en las aplicaciones para los prototipos con este *widget* de navegación. En la Figura 5.27 se ilustran las acciones que se deben llevar acabo para navegar a través de la taxonomía y completar la tarea.

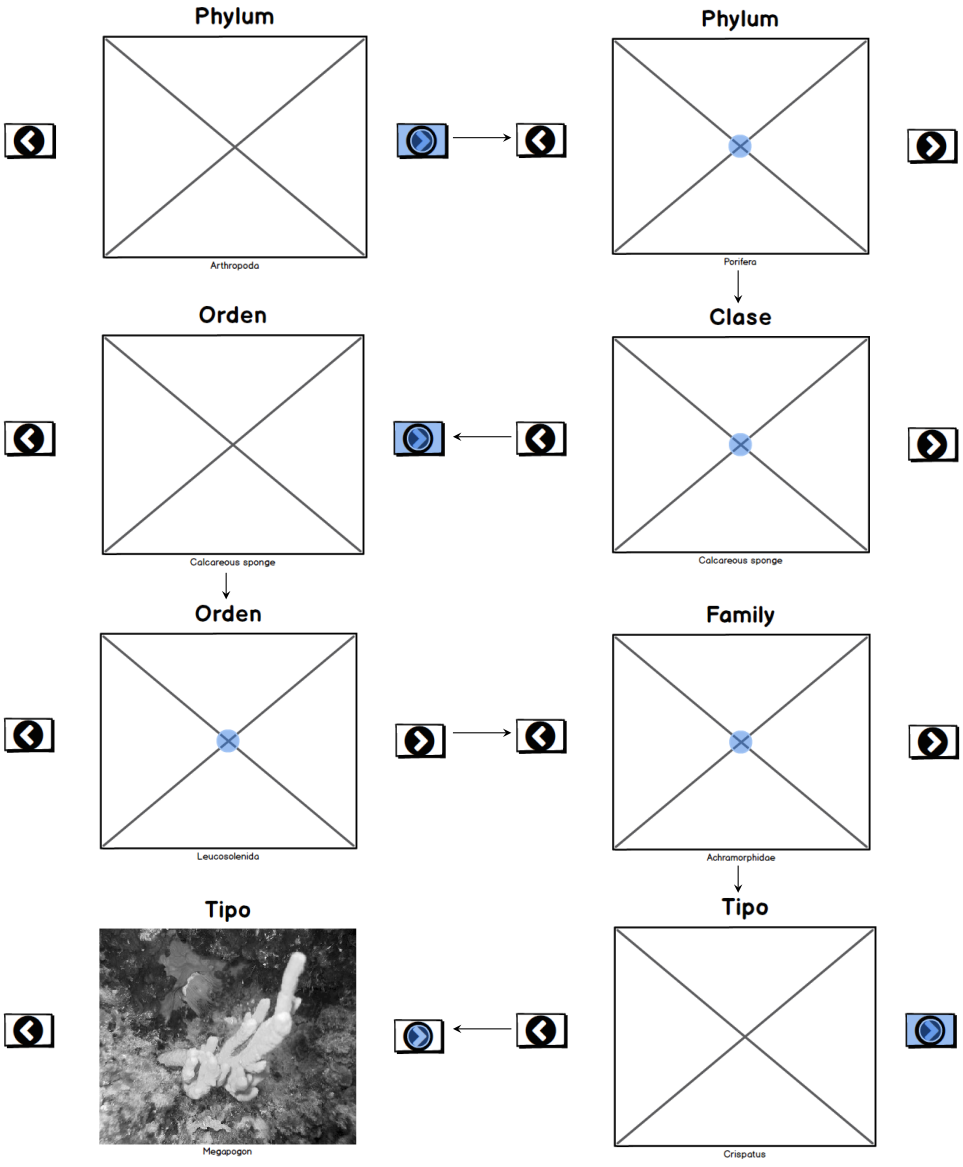


Figura 5.24. Acciones de galería en carrusel.

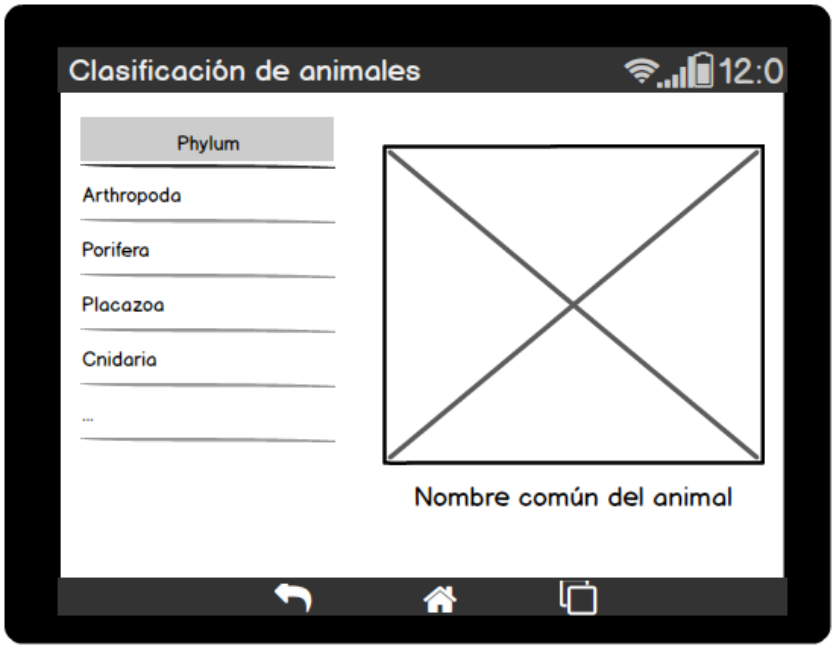


Figura 5.25. Esqueleto del la aplicación nativa con lista en páginas.

Figura 5.26. Esqueleto del la aplicación rich-client con lista en páginas.

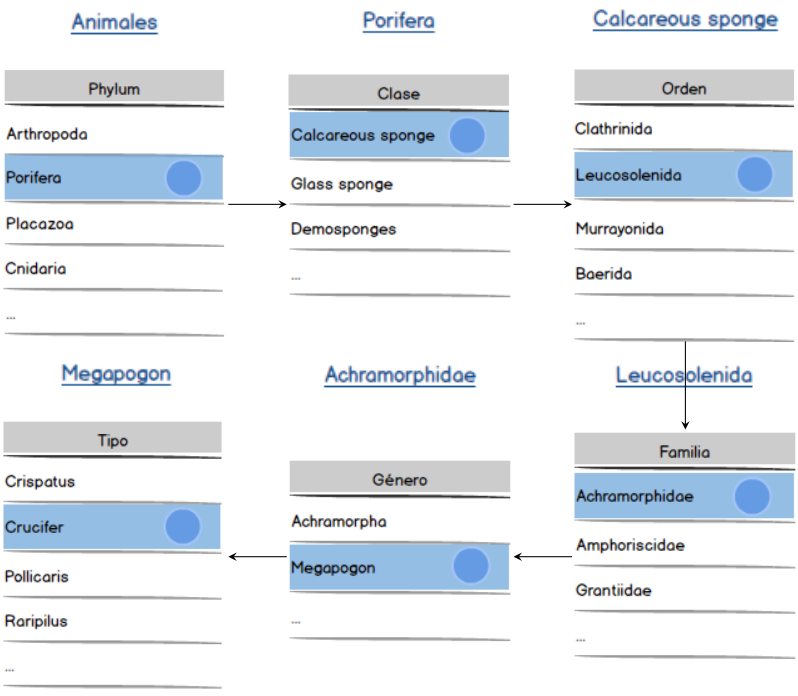


Figura 5.27. Acciones del widget de lista en páginas.

Al utilizar este tipo de *widget* es necesario contar con un árbol de navegación para saber en donde nos encontramos y permitir regresar a cualquier nivel de la taxonomía.

5.4. Resultados

Como se mencionó al inicio del capítulo se ejecutaron treinta pruebas para cada prototipo. El error absoluto en la medición de energía de los prototipos que se probaron oscilo entre 5 % y 10 %[2]. Los resultados de estos prototipos se compararon de acuerdo a la relación de equivalencia de *widgets* y al tipo de aplicaciones ejecutándose en máquina virtual o sin ella.

5.4.1. Resultados según el tipo de aplicación

En la Tabla 5.4 y 5.5 se muestran los resultados que se obtuvieron en consumo de energía y potencia respectivamente para cada prototipo sobre cada aplicación. En la Figura 5.28 y 5.29 se muestran las gráficas que comparan el comportamiento de energía y potencia de cada uno de los prototipos en las diferentes aplicaciones (nativas y *rich-Client*).

Widget	Nativa	Nativa con VM	*Rich-client*	*Rich-client* S.0-VM	*Rich-client* VM
Menú fijo	371.76	587.76	2334.03	2698.89	3690.15
Lista vertical	232.23	359.32	2277.12	2669.74	3523.25
Lista de páginas	114.91	204.72	1941.82	2375.40	3459.42
Galería en carrusel	623.76	986.28	3601.92	3635.46	5695.30
Lista de miniaturas	660.96	1032.72	3978.98	4382.97	5841.33
Galería en grid	656.64	970.32	3101.76	3363.09	4878.24

Tabla 5.4. Potencia (mW) de todos los los prototipos sobre las diferentes aplicaciones.

[2]El error absoluto esta calculado para cada prototipo como $err = \frac{\sum_{i=1}^{n} E_{Pi} - E_m}{n}$ donde E_{Pi} es la energía total de cada prueba, E_m es la energía promedio para cada prototipo y n es la cantidad de pruebas realizadas

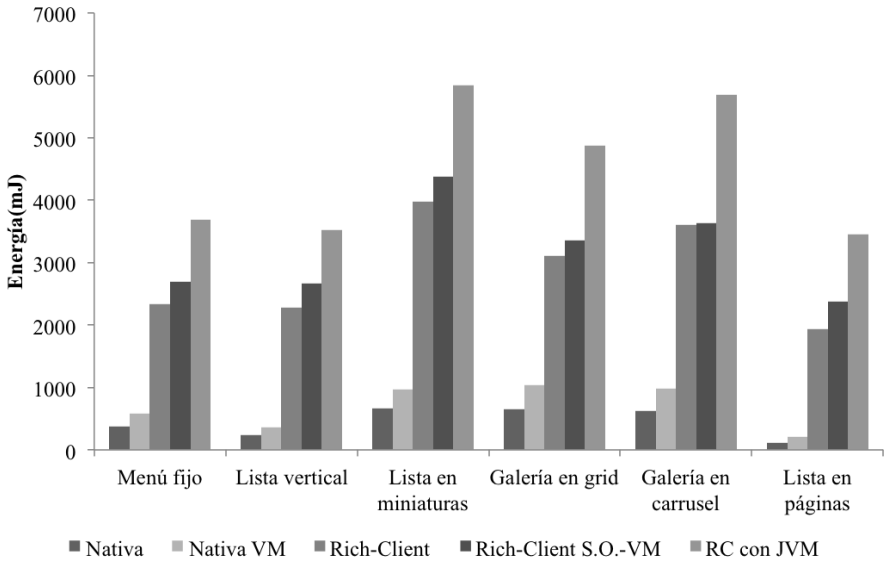


Figura 5.28. Gráfica comparativa general de consumo energético de los widgets sobre las aplicaciones nativas y rich-client.

Widget	Nativa	Nativa con VM	*Rich-client*	*Rich-client* S.0-VM	*Rich-client* VM
Menú fijo	31.49	42.43	118.66	133.49	153.76
Lista vertical	25.64	35.54	135.61	158.76	195.74
Lista de páginas	19.13	28.40	132.14	158.36	216.21
Galería en carrusel	37.21	54.99	122.84	125.36	203.40
Lista de miniaturas	42.36	67.41	165.79	182.62	243.39
Galería en grid	44.19	78.86	143.50	152.87	232.30

Tabla 5.5. Consumo de energía (mJ) de todos los los prototipos sobre las diferentes aplicaciones.

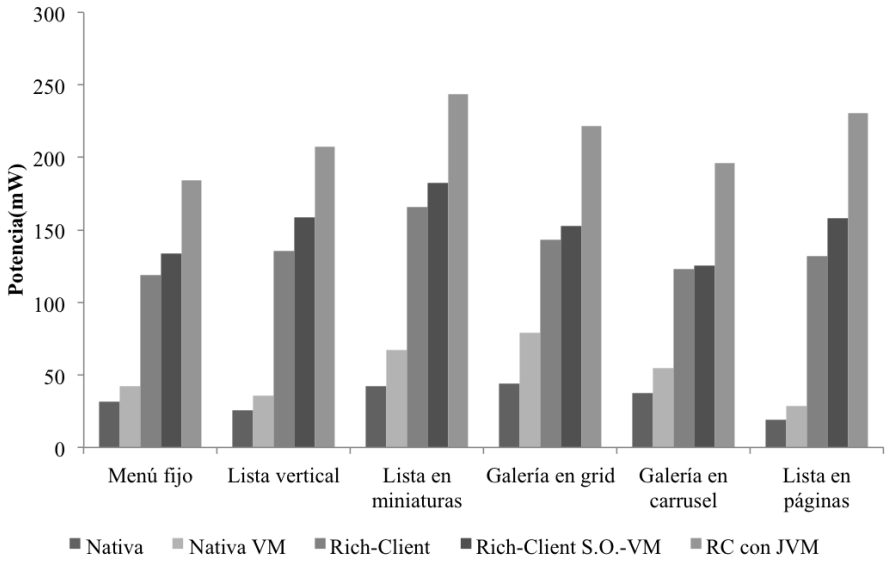


Figura 5.29. Gráfica comparativa general de potencia de los prototipos sobre las aplicaciones nativas y rich-client.

5.4.2. Resultados según la equivalencia de *widgets*

A continuación se muestran los resultados obtenidos de acuerdo a la relación de equivalencia (ver Sección 5.3). Estos *widgets* equivalentes son: menú fijo contra lista vertical, lista en miniaturas contra galería en *grid* y lista en páginas contra galería en carrusel.

Menú fijo y Lista vertical

En la Tabla 5.6 y 5.7 se muestran los resultados de energía y potencia respectivamente que se obtuvieron para el *widget* de menú fijo y el *widget* lista vertical. En la Figura 5.30 y 5.31 se muestran la gráficas que comparan el consumo de energía y potencia entre estos dos *widgets* de navegación.

Widget	Nativa	Nativa con VM	*Rich-client*	*Rich-client* S.O-VM	*Rich-client* VM
Menú fijo	371.76	587.76	2334.03	2698.89	3690.15
Lista vertical	232.23	359.32	2277.12	2669.74	3523.25

Tabla 5.6. Consumo de energía (mJ) en los prototipos del menú fijo y la lista vertical.

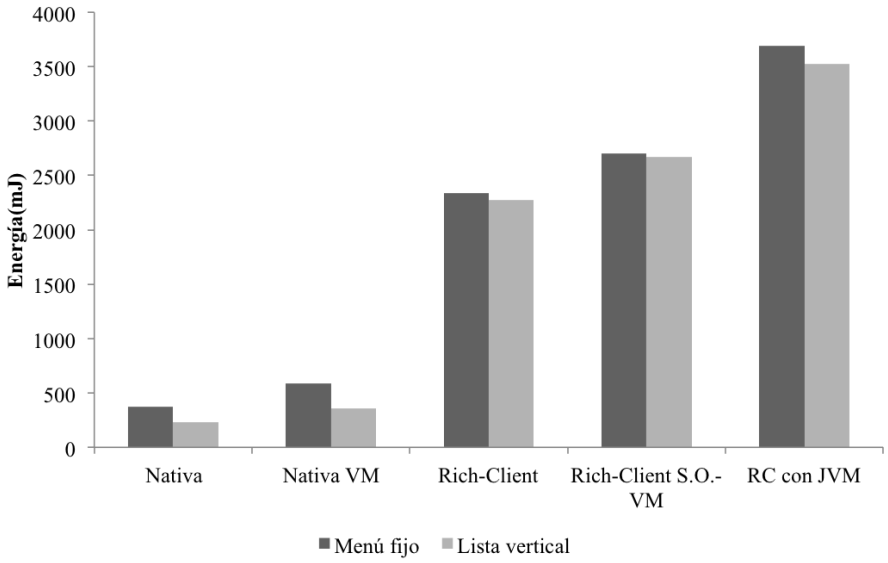


Figura 5.30. Gráfica comparativa de consumo energético entre menú fijo y lista vertical.

Widget	Nativa	Nativa con VM	*Rich-client*	*Rich-client* S.O-VM	*Rich-client* VM
Menú fijo	31.49	42.43	118.66	133.49	153.76
Lista vertical	25.64	35.54	135.61	158.76	195.74

Tabla 5.7. Potencia (mW) en los prototipos del menú fijo y la lista vertical.

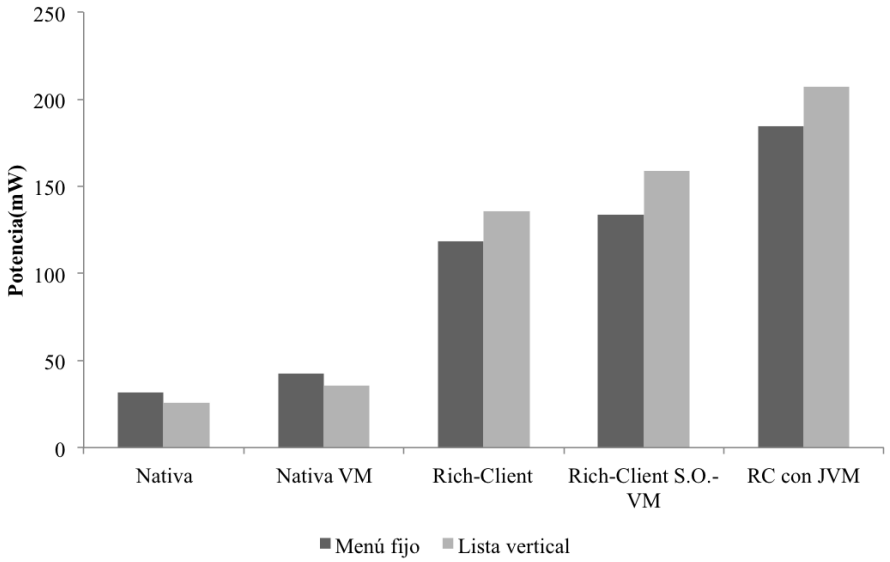


Figura 5.31. Gráfica comparativa de potencia entre menú fijo y lista vertical.

Lista de páginas y Galería en carrusel

En la Tabla 5.8 y 5.9 se muestra los resultados que se obtuvieron en consumo de energía y potencia para los *widgets* de lista de páginas y galería en carrusel. En la Figura 5.32 y 5.33 se muestran las gráficas que comparan el consumo de energía y potencia entre estos dos *widgets* de navegación.

Widget	Nativa	Nativa con VM	*Rich-client*	*Rich-client* S.O-VM	*Rich-client* VM
Lista de páginas	114.91	204.72	1941.82	2375.40	3459.42
Galería en carrusel	623.76	986.28	3601.92	3635.46	5695.30

Tabla 5.8. Consumo de energía (mJ) en los prototipos de la lista de páginas y la galería en carrusel.

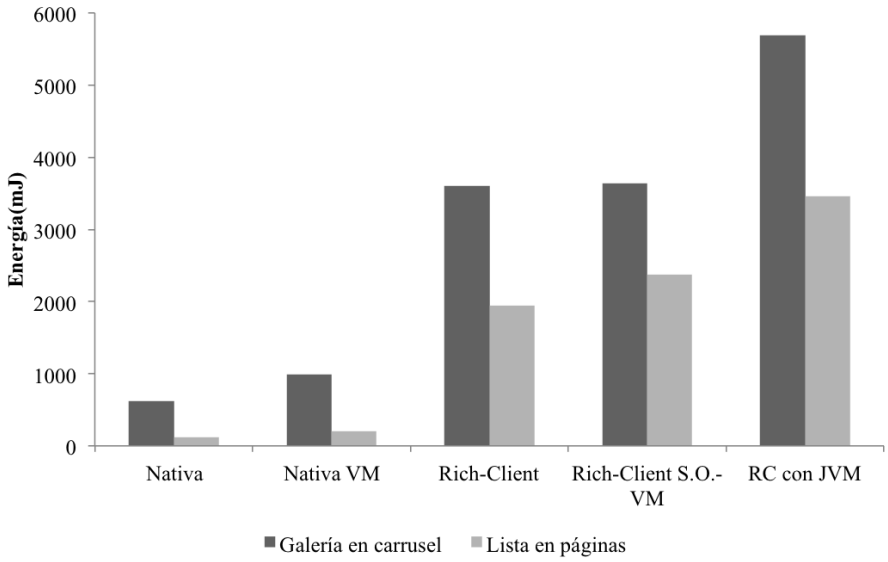


Figura 5.32. Gráfica comparativa de consumo energético entre la lista de páginas y la galería en carrusel.

Widget	Nativa	Nativa con VM	*Rich-client*	*Rich-client* S.O-VM	*Rich-client* VM
Lista de páginas	19.13	28.40	132.14	158.36	216.21
Galería en carrusel	37.21	54.99	122.84	125.36	203.40

Tabla 5.9. Potencia (mW) en los prototipos de la lista de páginas y la galería en carrusel.

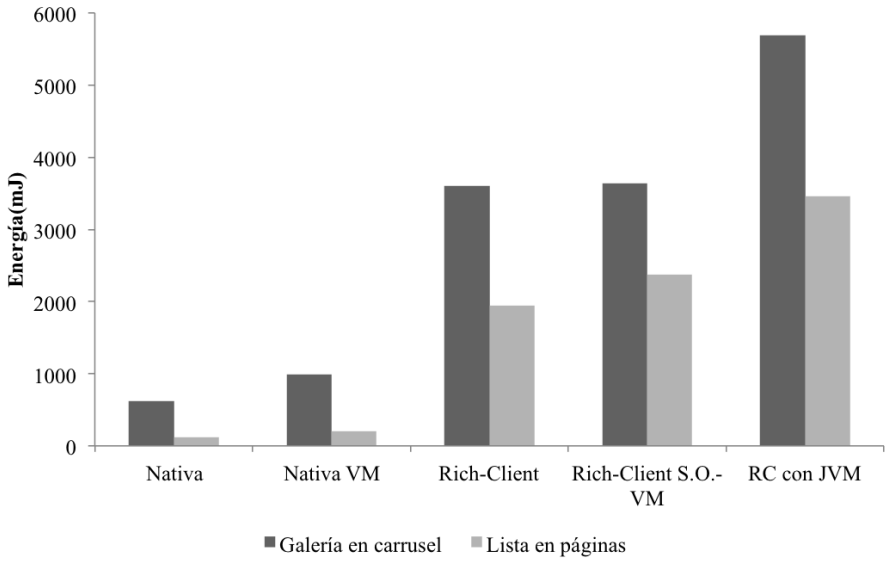


Figura 5.33. Gráfica comparativa de potencia entre la lista de páginas y la galería en carrusel.

Lista de miniaturas y Galería en *grid*

En la Tabla 5.10 y 5.11 se muestra los resultados que se obtuvieron en consumo de energía y potencia para los *widgets* de lista de miniaturas y galería en grid. En la Figura 5.34 y 5.35 se muestran las gráficas que comparan estos dos *widgets* de navegación.

Widget	Nativa	Nativa con VM	*Rich-client*	*Rich-client* S.O-VM	*Rich-client* VM
Lista de miniaturas	660.96	1032.72	3978.98	4382.97	5841.33
Galería en grid	656.64	970.32	3101.76	3363.09	4878.24

Tabla 5.10. Consumo de energía (mJ) en los prototipos de la lista de miniaturas y la lista en grid.

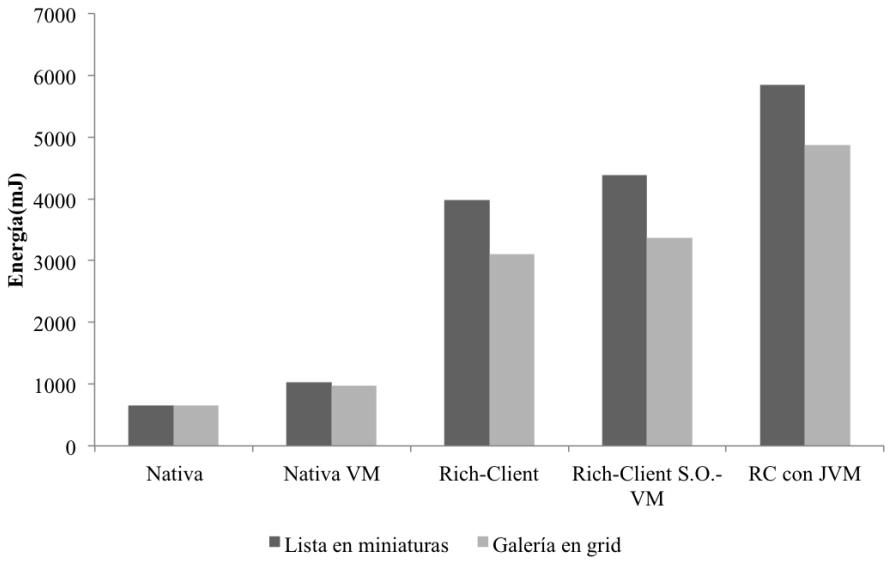


Figura 5.34. Gráfica comparativa de consumo de energía entre la lista de miniaturas y la lista en grid.

Widget	Nativa	Nativa con VM	*Rich-clientt*	*Rich-client* S.O-VM	*Rich-client* VM
Lista de miniaturas	42.36	67.41	165.79	182.62	243.39
Galería en grid	44.19	78.86	143.50	152.87	232.30

Tabla 5.11. Potencia (mW) en los prototipos de la lista de miniaturas y la lista en grid.

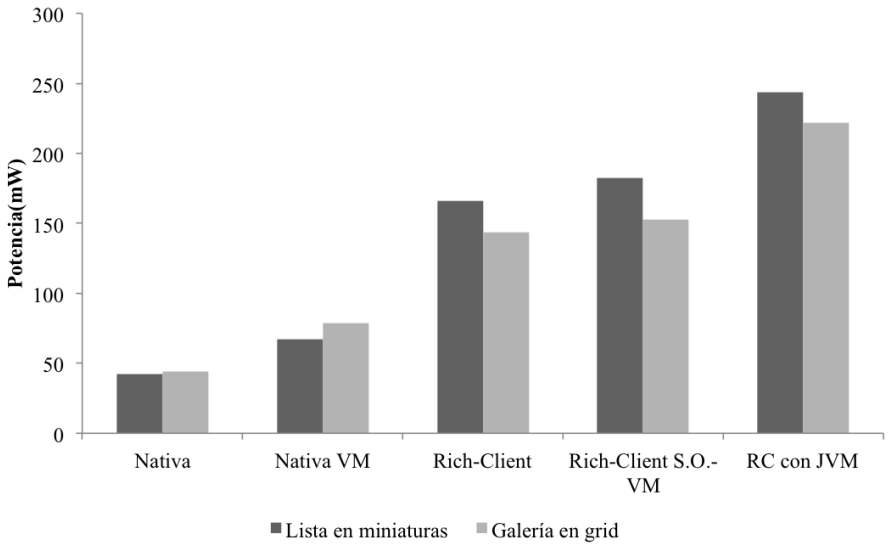


Figura 5.35. Gráfica comparativa de potencia entre la lista de miniaturas y la lista en grid.

5.5. Discusión

Para obtener los resultados se trabajó en un ambiente controlado, es decir con una carga constante en batería sin variaciones y la iluminación de la pantalla en 100 %. En base a esto, en muchos de los resultados que se obtuvieron se observo que el comportamiento de la energía con respecto a la potencia sin importar si se ejecuta en una máquina virtual o no se mantuvo en las aplicaciones nativas, mientras que en las aplicaciones *rich-client* el comportamiento varió. Esto se debe principalmente por que los prototipos como lista vertical o lista en páginas utilizan meno unidades de tiempo, por lo que aunque utilizan más potencia su tiempo de ejecución es menor y por lo tanto utilizan menos energía.

En las aplicaciones que trabajan con máquina virtual (JVM) consumen al rededor de un 0.5 a 0.8 veces más de energía que las aplicaciones que no la utilizan. En consumo de potencia de las aplicaciones que trabajan con la JVM es de 0.3 a un 0.6 veces más de las que no la utilizan. Este comportamiento se debe a que la máquina virtual realiza una traducción hacia la máquina que la esta ejecutando, es decir por cada interacción o despliegue en la pantalla tiene que realizar dicha tarea.

Se aprecia que las aplicaciones que se ejecutan como *rich-Client* requieren de 5 a 15 veces más de energía que las aplicaciones nativas, independientemente del ambiente que se este utilizando (con máquina virtual o sin ella) y el tipo de *widget* que se esté manejando. Esto depende en gran medida de la cantidad de instrucciones que pueda ejecutar el navegador y, además que tan rápido funciona sobre el sistema en el cual se encuentra instalado, por lo cual se requiere de más tiempo de ejecución. En consumo de potencia de las aplicaciones *rich-client* con respecto a las nativas es de 3 a 6 veces más. Este comportamiento se debe principalmente al tiempo de ejecución del prototipo, es decir que un prototipo puede consumir más energía y menos potencia o consumir menos energía y utilizar más potencia.

En general, los prototipos que no utilizan imágenes consumen menos energía ya que sólo despliega texto en pantalla y por lo tanto son más rápidos para completar una tarea. Sin embargo, se observó que el consumo de potencia, en particular en los prototipos que se ejecutaron como *rich-client*, pueden consumir menos energía, pero más potencia. Esto se debe a que algunos prototipos utilizan más el *scroll*, lo cual hace que haya una mayor interacción.

Al realizar las comparaciones de energía en base a la equivalencia de *widgets* se tienen los siguientes resultados.

Menú fijo y lista vertical

De los experimentos realizados se observó que el menú fijo consume alrededor del 0.62 veces más de energía en aplicaciones nativas y un 0.04 veces más para aplicaciones *rich-client* que el prototipo en lista vertical. Sin embargo, en el consumo de potencia el comportamiento en las aplicaciones nativas el menú fijo consume alrededor del 0.18 veces más que la lista vertical, en cambio en las aplicaciones *rich-client* la lista vertical consume más potencia, al rededor de un 0.2 veces.

El comportamiento en consumo de potencia, particularmente en las aplicaciones *rich-client*, lista vertical necesita más potencia, se debe a que el uso del *scroll* aumenta de acuerdo a su diseño (ver la Figura 5.14), por lo que la interacción en este *widget* dura más. Sin embargo, en todos los prototipos sin importar el tipo de aplicación el menú fijo consume más energía por que requiere de más acciones para completar la tarea, es decir el usuario tiene que interaccionar con este *widget* durante más tiempo.

Lista de páginas y galería en carrusel

De acuerdo a los experimentos realizados, la galería en carrusel consume alrededor de 4.12 veces más de energía en aplicaciones nativas y un 0.75

veces más para aplicaciones *rich-client* que el prototipo en lista en páginas. Sin embargo, en el consumo de potencia el comportamiento en aplicaciones nativas la galería en carrusel consume al rededor del 0.9 veces, mientras que en las aplicaciones *rich-client* la galería en carrusel y lista en páginas tienen un comportamiento similar, sólo se diferencia por 0.06 veces más que consume lista en páginas.

Este comportamiento en energía se debe a que el *widget* de galería en carrusel trabaja con imágenes a diferencia de lista de páginas, por lo que al actualizar la pantalla se requiere más memoria en el *framebuffer* y la pantalla requiere desplegar varios tonos de colores. También se puede observar en la Tabla 5.3 que el *widget* de lista en páginas realiza 6 acciones para completar la tarea mientras que galería en carrusel depende de la cantidad de veces que se clic hacia la izquierda o la derecha. lo que requiere más tiempo de interacción según el usuario lo requiera. Sin embargo en consumo de potencia en las aplicaciones nativas mantienen el comportamiento de acuerdo a la energía consumida, mientras que en las aplicaciones *rich-client* el consumo de potencia es la lista en páginas ligeramente superior a la galería en carrusel. Esto se debe a que, en algunas ocasiones, se requiere utilizar el *scroll* para ver los elementos de la lista.

Lista en miniaturas y galería en *grid*

Como se observa en los prototipos de estos *widgets* ambos utilizan imágenes. De acuerdo a las pruebas realizadas la lista en miniatura consume alrededor de 0.05 veces más de energía en aplicaciones nativas y 0.20 veces más en aplicaciones *rich-client* que la lista en miniaturas. En el consumo de potencia, el comportamiento de galería en *grid* en las aplicaciones nativas es de 0.05 veces más con respecto a la lista en miniatura, mientras que en las aplicaciones *rich-client* la lista en miniaturas consume 0.13 veces más de potencia.

La lista en miniatura consume más energía debido a su diseño (ver la

Figura 5.16), ya que a pesar que no tiene que desplegar varias imágenes al mismo tiempo el uso del *scroll* aumenta lo que provoca que la energía aumente. Sin embargo, en el consumo de potencia en las aplicaciones nativas el *widget* de lista en miniatura consume un poco menos de potencia debidp a que la pantalla no tiene que desplegar varias imágenes a la vez.

CAPÍTULO 6

Conclusiones y trabajo a futuro

6.1. Conclusiones

Existen varias técnicas de ahorro de energía que permiten a los dispositivos móviles alargar el tiempo de carga de la batería. Estás técnicas son: el planificador, la optimización de la máquina virtual, el *offloading*, el *offlining*, el uso de red, el monitor de la batería, el procesamiento secuencial y el enfoque de diseño. Sin embargo, la mayoría de éstas técnicas se encuentran enfocadas en la manera de como se ejecutan los procesos y las tareas en el CPU, sin tomar en cuenta que la pantalla es el principal componente que consumen demasiada energía y potencia, ya que además de desplegar la información, esta requiere del *framebuffer* para actualizar constantemente la información.

Para permitir al usuario interaccionar con el dispositivo móvil debe de realizarse a través de una interfaz gráfica de usuario (GUI) que se despliega en la pantalla. La GUI esta compuesta de diferentes componentes gráficos llamados *widgets* como: los de navegación, los de entrada, los de control y los de presentación, que permiten al usuario realizar las diversas tareas que requiera dependiendo del tipo de aplicación que utilice. Sin embargo, siempre es necesario navegar a través de una aplicación con GUI, por lo que se requiere siempre de los *widgets* de navegación. Esto los vuelve indispensables e importantes para que el usuario pueda desarrollar las tareas que requiera permitiendo a la GUI cambiar de estados.

Estos *widgets* de navegación pueden clasificarse no sólo por funcionalidad, sino también por usabilidad, es decir tener una equivalencia tomando en cuenta dichos factores. Esta equivalencia permite al desarrollador realizar diseños flexibles de su aplicación manteniendo su la funcionalidad y usabilidad.

Las aplicaciones móviles pueden ser de tipo nativas, clientes o *rich-Client*. Sin embargo, las aplicaciones móviles deben ser capaces de distinguir tanto los componentes del dispositivo móvil así como las características del sistema operativo. Por lo tanto, es recomendado implementar aplicaciones nativas que son desarrolladas de manera exclusiva para un sistema operativo en específico, o bien de tipo *rich-client* que son aplicaciones multiplataforma que se ejecutan sobre un navegador web y que a través de código HTML5 y JavaScript permita conocer las características del dispositivo. Las aplicaciones *rich-client* están sujetas a la restricción de que el navegador sea capaz de ejecutar dicho código.

Por otra parte, medir el consumo de energía de un dispositivo móvil se puede realizar mediante herramientas que se comunican con el ACPI y que utilizan diversas técnicas como minería de datos para estimar el consumo de energía de los componentes desde el consumo total de energía del sistema. Sin embargo, la medición con este tipo de herramientas no es muy exacta, esto se debe a que la energía sólo se estima. Por otro lado, también se puede medir la energía mediante el *hardware*. Se pueden utilizar herramientas como el osciloscopio o el multímetro que permiten monitorear la corriente utilizada.

Se diseñaron prototipos que permiten evaluar el consumo de energía de algunos *widgets* de navegación. Estos prototipos estresan los componentes que utiliza la GUI para realizar la comparación entre la aplicación nativa y *rich-Client*. También se diseñaron para que se ejecuten con máquina virtual y sin ella.

Con los resultados que se obtuvieron se concluye que las aplicaciones que se ejecutan sobre una máquina virtual consumen al rededor de un 50 %a un 80 % más de energía y al rededor de 30 % a un 60 % más de potencia que

las que no la utilizan. Principalmente se debe al que la máquina virtual debe de realizar la traducción de su código al código del dispositivo móvil en el que se este ejecutando. Por lo tanto es recomendable desarrollar aplicaciones nativas para evitar un gran impacto en la batería.

Al realizar la comparación de energía entre las aplicaciones nativas y las *rich-Client* se tiene que las de tipo *rich-Client* consume alrededor de cinco a quince veces más de energía y de tres a seis veces más de energía. Este comportamiento se da por diversos factores tales como: la optimización del navegador para el S.O. del dispositivo móvil en el que se este ejecutando y la cantidad de instrucciones que procesa por segundo.

Se encontró que al tener *widgets* equivalentes, es decir misma funcionalidad y usabilidad, existen unos que consumen más energía que otros. Por ejemplo, si el desarrollador desea trabajar con un menú fijo, puede optar por trabajar con una lista vertical en una aplicación *rich-client*. Sin embargo, el comportamiento de la potencia no es el mismo que en la energía, principalmente en las aplicaciones *rich-client*. Especialmente se debe tener cuidado con los *widgets* que utilizan el *scroll* para desplazarse a través de él, ya que el usó excesivo de este elemento puede perjudicar en gran medida en el consumo de potencia.

Con los resultados obtenidos, elegir que tipo de *widget* utilizar, se debe considerar los atributos que lo conforman como son la cantidad de elementos, las imáguenes que requiere (tomando en cuenta el color, el tipo y el tamaño) y como se desea mostrar la información, ya que esto puede tener un impacto directo en el consumo de energía.

Por lo tanto, el desarrollador puede realizar una comparativa de consumo de energía y de potencia entre los *widgets* equivalentes que permitirá a su aplicación tener la misma funcionalidad y usabilidad además de manejar una forma razonable la batería. De esta forma, el desarrollador será capaz considerar como otro indicador el consumo de energía para el diseño de la GUI.

6.2. Trabajo a futuro

Este trabajo es el inicio del estudio de consumo de energía en las interfaces gráficas de usuarios utilizando los *widgets* de navegación. Por lo cual, queda mucho trabajo por desarrollar para seguir ampliando el estado del arte y ofrecer un indicador que permita al desarrollador considerar el uso de la batería en el diseño de sus interfaces gráficas de usuario, para ello se requiere:

- Tener una comparativa más completa de *widgets* equivalentes para realizar pruebas de consumo de energía entre ellos.
- Establecer *benchmark* que estandaricen las pruebas entre las interfaces gráficas de usuario.
- Desarrollar los prototipos a las plataformas móviles más comunes que funcionen con máquina virtual y sin ella, por ejemplo Andorid o iOS.
- Desarrollar una herramienta que permita medir el consumo de energía de la pantalla y las tareas que asociadas al despliegue de la información (el uso del *framebuffer* y el procesamiento).
- Desarrollar modelos que permitan tener un análisis de consumo de energía y potencia.
- Realizar las pruebas con diferentes niveles de carga de la batería.

Bibliografía

[1] Abogharaf, Abdulhakim, Rajesh Palit, Kshirasagar Naik y Ajit Singh: *A methodology for energy performance testing of smartphone applications*. En *Automation of Software Test (AST), 2012 7th International Workshop on*, páginas 110–116. IEEE, 2012.

[2] Berl, Andreas, Erol Gelenbe, Marco Di Girolamo, Giovanni Giuliani, Hermann De Meer, Minh Quan Dang y Kostas Pentikousis: *Energy-efficient cloud computing.* The computer journal, 53(7):1045–1051, 2010.

[3] Bircher, W Lloyd, Madhavi Valluri, Jason Law y Lizy K John: *Runtime identification of microprocessor energy saving opportunities.* En *Low Power Electronics and Design, 2005. ISLPED'05. Proceedings of the 2005 International Symposium on*, páginas 275–280. IEEE, 2005.

[4] Carroll, Aaron y Gernot Heiser: *An Analysis of Power Consumption in a Smartphone.* En *USENIX annual technical conference*, páginas 271–285, Massachusetts, United States, June 2010.

[5] Carroll, Aaron y Gernot Heiser: *The systems hacker's guide to the galaxy energy usage in a modern smartphone.* En *Proceedings of the 4th Asia-Pacific Workshop on Systems*, página 5. ACM, 2013.

[6] Carroll, Aaron y Gernot Heiser: *Unifying DVFS and Offlining in Mobile Multicores.* En *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Berlin, Germany, April 2014.

[7] Cuervo, Eduardo, Aruna Balasubramanian, Dae ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra y Paramvir Bahl: *MAUI: making smartphones last longer with code offload.* En *Proceedings of the 8th international conference on Mobile systems, applications, and services*, páginas 49–62. ACM, 2010.

[8] D'Ambrosio, Salvatore, Salvatore De Pasquale, Gerardo Iannone, Delfina Malandrino, Alberto Negro, Giovanni Patimo, Andrea Petta, Vittorio Scarano, Luigi Serra y Raffaele Spinelli: *Mobile phone batteries draining: is Green Web Browsing the solution?* En *Green Computing Conference (IGCC), 2014 International*, páginas 1–10. IEEE, 2014.

[9] Dinh, Hoang T, Chonho Lee, Dusit Niyato y Ping Wang: *A survey of mobile cloud computing: architecture, applications, and approaches.* Wireless communications and mobile computing, 13(18):1587–1611, 2013.

[10] Dolezal, Jakub y Zdenek Becvar: *Methodology and tool for energy consumption modeling of mobile devices.* En *Wireless Communications and Networking Conference Workshops (WCNCW), 2014 IEEE*, páginas 34–39. IEEE, 2014.

[11] Emmanuel, Vázquez Ceballos Dario: *Estudio de equivalencia de widgets para dispositivos móviles.* Tesis de Licenciatura, ESCOM-IPN, Por presentarse, 2015.

[12] Galitz, Wilbert O: *The essential guide to user interface design: an introduction to GUI design principles and techniques.* John Wiley & Sons, 2007.

[13] Heikkinen, Mikko VJ, Jukka K Nurminen, Timo Smura y Heikki Hämmäinen: *Energy efficiency of mobile handsets: Measuring user attitudes and behavior.* Telematics and Informatics, 29(4):387–399, Elsevier, 2012.

[14] Helander, Martin G: *Handbook of human-computer interaction*. Elsevier, 2014.

[15] Hong, Yu Ju, Karthik Kumar y Yung Hsiang Lu: *Energy efficient content-based image retrieval for mobile systems*. En *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, páginas 1673–1676. IEEE, 2009.

[16] Hoober, Steven y Eric Berkman: *Designing mobile interfaces*. O'Reilly Media, Inc., 2011.

[17] Hou, ZQR, Jong Ching Chuen y Andreas Herkersdorf: *Apps-usage driven energy management for multicore mobile computing systems*. En *Integrated Circuits (ISIC), 2014 14th International Symposium on*, páginas 472–475. IEEE, 2014.

[18] Kemp, Roelof, Nicholas Palmer, Thilo Kielmann y Henri Bal: *Cuckoo: a computation offloading framework for smartphones*. En *Mobile Computing, Applications, and Services*, páginas 59–79. Springer, 2012.

[19] Korhonen, Kaisa: *Predicting mobile device battery life*. Tesis de Licenciatura, School of Electrical Engineering, Aalto University, 2011.

[20] Kreiman, Edward y Emil C Lupu: *Using Learning to Predict and Optimise Power Consumption in Mobile Devices.*, 2010.

[21] Kumar, Karthik, Jibang Liu, Yung Hsiang Lu y Bharat Bhargava: *A survey of computation offloading for mobile systems*. Mobile Networks and Applications, 18(1):129–140, 2013.

[22] Kumar, Karthik y Yung Hsiang Lu: *Cloud computing for mobile users: Can offloading computation save energy?* Computer, (4):51–56, 2010.

[23] Kundu, Tapas Kumar y Kolin Paul: *Improving Android performance and energy efficiency*. En *VLSI Design (VLSI Design), 2011 24th International Conference on*, páginas 256–261. IEEE, 2011.

[24] Lee, Valentino, Heather Schneider y Robbie Schell: *Mobile applications: architecture, design, and development.* Prentice Hall PTR, 2004.

[25] Liu, Hao, Yaoxue Zhang y Yuezhi Zhou: *Tailtheft: leveraging the wasted time for saving energy in cellular communications.* En *Proceedings of the sixth international workshop on MobiArch*, páginas 31–36. ACM, 2011.

[26] Looga, V., Y. Xiao, Z. Ou y A. Yla-Jaaski: *Exploiting traffic scheduling mechanisms to reduce transmission cost on mobile devices.* En *Wireless Communications and Networking Conference (WCNC), Paris, France, Abril 1-4*, páginas 1766–1770. IEEE, 2012.

[27] Love, Steve: *Understanding mobile human-computer interaction.* Butterworth-Heinemann, 2005.

[28] Luo, Lu: *Efficient Interactions with Mobile Systems.* Tesis de Doctorado, School of Computer Science Institute for Software Research, Carnegie Mellon University, Institute for Software Research, Carnegie Mellon University, 2008.

[29] Meier, J.D., Alex Homer, David Hill, Jason Taylor, Prashant Bansode, Lonnie Wall, Rob Jr Boucher y Akshay Bogawat: *Mobile Application Architecture Guide.* Microsoft Corporation, 2008.

[30] Miettinen, Antti P y Jukka K Nurminen: *Energy efficiency of mobile clients in cloud computing.* En *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, páginas 4–4. USENIX Association, 2010.

[31] Mittal, Radhika, Aman Kansal y Ranveer Chandra: *Empowering developers to estimate app energy consumption.* En *Proceedings of the 18th annual international conference on Mobile computing and networking*, páginas 317–328. ACM, 2012.

[32] Pathak, Abhinav, Y Charlie Hu y Ming Zhang: *Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof.* En *Proceedings of the 7th ACM european conference on Computer Systems*, páginas 29–42. ACM, 2012.

[33] Paul, Kolin y Tapas Kumar Kundu: *Android on mobile devices: An energy perspective.* En *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, páginas 2421–2426. IEEE, 2010.

[34] Perrucci, Gian Paolo, Frank HP Fitzek y Jörg Widmer: *Survey on energy consumption entities on the smartphone platform.* En *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, páginas 1–6, Budapest, Hungary, May 2011. IEEE.

[35] Rahmati, Ahmad y Lin Zhong: *Human–battery interaction on mobile phones.* Pervasive and Mobile Computing, 5(5):465–477, 2009.

[36] Ricardo Isidro, Ramírez, Meneses Viveros A y Hernandez Rubio E: *Energy consumption model over parallel programs implemented on multicore architectures.* International Journal of Advanced Computer Science and Applications, 6(6):252–259, 2015.

[37] Rice, Andrew y Simon Hay: *Measuring mobile phone energy consumption for 802.11 wireless networking.* Pervasive and Mobile Computing, 6(6):593–606, 2010.

[38] Ristanovic, Nikodin, Jean Yves Le Boudec, Augustin Chaintreau y Vijay Erramilli: *Energy efficient offloading of 3G networks.* En *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, páginas 202–211. IEEE, 2011.

[39] Robinson, Stuart: *Cellphone energy gap: Desperately seeking solutions*. Strategy Analytics, 2009. `https://www.strategyanalytics.com/default.aspx?mod=reportabstractviewer&a0=4645`.

[40] Tarkoma, Sasu, Matti Siekkinen, Eemil Lagerspetz y Yu Xiao: *Smartphone energy consumption: modeling and optimization*. Cambridge University Press, 2014.

[41] Thiagarajan, Narendran, Gaurav Aggarwal, Angela Nicoara, Dan Boneh y Jatinder Pal Singh: *Who killed my battery?: analyzing mobile browser energy consumption*. En *Proceedings of the 21st international conference on World Wide Web*, páginas 41–50. ACM, 2012.

[42] Tiffany, Rob: *Mobile Application Architecture Guide*. Microsoft Corporation, 2008.

[43] Tseng, Po Hsien, Pi Cheng Hsiu, Chin Chiang Pan y Tei Wei Kuo: *User-centric energy-efficient scheduling on multi-core mobile devices*. En *Proceedings of the 51st Annual Design Automation Conference*, páginas 1–6. ACM, 2014.

[44] Vieira, Agata, Daniel Debastiani, Luciano Agostini, Felipe Marques y Julio CB Mattos: *Performance and Energy Consumption Analysis of Embedded Applications Based on Android Platform*. En *Computing System Engineering (SBESC), 2012 Brazilian Symposium on*, páginas 59–64. IEEE, 2012.

[45] Viveros, Amilcar Meneses, Erika Hernández Rubio y Dario Emmanuel Vázquez Ceballos: *Equivalence of Navigation Widgets for Mobile Platforms*. En *Design, User Experience, and Usability. User Experience Design for Diverse Interaction Platforms and Environments*, páginas 269–278. Springer, 2014.

[46] Wang, Le y Jukka Manner: *Evaluation of data compression for energy-aware communication in mobile networks.* En *Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009. CyberC'09. International Conference on*, páginas 69–76. IEEE, 2009.

[47] Wolski, Rich, Selim Gurun, Chandra Krintz y Dan Nurmi: *Using bandwidth data to make computation offloading decisions.* En *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, páginas 1–8. IEEE, 2008.

[48] Wroblewski, Luke: *Mobile first: Préface de Jeffrey Zeldmann.* Editions Eyrolles, 2012.

[49] Yoon, Chanmin, Dongwon Kim, Wonwoo Jung, Chulkoo Kang y Hojung Cha: *AppScope: Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring.* En *USENIX Annual Technical Conference*, páginas 387–400, 2012.

[50] Yuan, Wanghong y Klara Nahrstedt: *Energy-efficient soft real-time CPU scheduling for mobile multimedia systems.* ACM SIGOPS Operating Systems Review, 37(5):149–163, 2003.

[51] Zhang, Lide, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P Dick, Zhuoqing Morley Mao y Lei Yang: *Accurate online power estimation and automatic battery behavior based power model generation for smartphones.* En *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, páginas 105–114. ACM, 2010.