Mecanismos de Reconfiguración Eco-eficiente de Código en Aplicaciones Móviles Android

Angel Canete, José Miguel Horcas, y Lidia Fuentes

Universidad de Málaga, Andalucía Tech, Spain
{angelcv,horcas,lff}@lcc.uma.es, http://caosd.lcc.uma.es/

Resumen Los dispositivos móviles ofrecen cada vez mayores prestaciones a costa de un mayor consumo energético. La energía consumida por un móvil no sólo depende de las aplicaciones en sí, sino también de las interacciones del usuario con la aplicación. Si un recurso no está siendo utilizado por la aplicación, no debería estar consumiendo energía. En este artículo se presenta un modelo de adaptación de aplicaciones móviles al contexto del usuario con el objetivo de reducir el consumo energético de las aplicaciones. Se desarrollan y evalúan cuatro implementaciones diferentes de la propuesta en busca del mecanismo de reconfiguración más eficiente energéticamente.

Keywords: Adaptabilidad, Android, Dispositivos móviles, Eco-Eficiente, Energía, Reconfiguración dinámica

1. Introducción

El uso de los teléfonos inteligentes cada vez supera más al de los ordenadores personales. Para el año 2020, se prevé que el 70 % de la población mundial dispondrá de un *Smartphone* [7]. Existe la necesidad, cada vez mayor, por parte de los usuarios de mantenerse conectados en todo momento, pudiendo acceder a la información y a los servicios que deseen desde cualquier lugar. Sin embargo, con el aumento de las prestaciones que ofrecen estos dispositivos, mayor es su consumo energético (mejores procesadores, mayor ancho de banda, pantallas mayores y con mejor resolución, etc). Dicho consumo de energía afecta de manera directa al tiempo de duración de las baterías. Esto hace que el usuario deba cargar su teléfono diariamente, llegando en ocasiones a la necesidad de utilizar baterías externas para mantener el dispositivo encendido a lo largo del día. Pese a ser el hardware del teléfono el que consume esta energía, es el software el encargado de gestionar dichos recursos y, por tanto, influye enormemente en el consumo de energía del dispositivo.

La energía que consume una aplicación no sólo depende del código desplegado, sino también de las interacciones del usuario, ya que es el usuario quien finalmente hace uso de las funcionalidades que proporciona la aplicación. Los perfiles de los usuarios suelen ser muy diversos: algunos de ellos utilizan toda la funcionalidad proporcionada por la aplicación, mientras que otros usuarios sólo utilizan una mínima parte de los servicios. Lo ideal es que si el usuario no está

utilizando cierto recurso (e.g., la pantalla), ese recurso no consuma energía. De la misma forma, la funcionalidad llevada a cabo por la aplicación debería ajustarse al estado en que se encuentra el dispositivo, como su nivel de batería, ubicación actual o tipo de red al que se encuentra conectado, entre otros muchos factores de configuración posibles.

Para poder reducir el consumo energético en función del contexto, es necesario adaptar dinámicamente las aplicaciones. Los mecanismos de adaptación existentes que buscan reducir el consumo energético en dispositivos móviles normalmente no tienen en cuenta el comportamiento del usuario con la aplicación [1,2,3,5,9]. En particular, para el sistema Android, los mecanismos de reconfiguración de código son muy dependientes de las diferentes versiones del sistema operativo, quedando obsoletos al poco tiempo [5]. Además, cualquier mecanismo de reconfiguración introduce una sobrecarga que evidentemente conlleva un consumo de energía, por lo tanto el objetivo es explorar varios mecanismos de reconfiguración para ver cuál de ellos consume menos energía.

En este artículo se propone un modelo de adaptación basado en el contexto para aplicaciones Android, que permite reconfigurar la funcionalidad de una aplicación para mejorar su consumo energético en función del perfil de uso del usuario y el estado del dispositivo. Siguiendo este modelo, se han implementado cuatro soluciones de reconfiguración de código en busca del mecanismo de reconfiguración más eco-eficiente (aquel que consume menos energía), y que sea soportado por la gran variedad de versiones del sistema Android existentes en la actualidad. Se han evaluado y comparado las diferentes soluciones con un caso de estudio, demostrando que nuestra propuesta de reconfiguración mejora el consumo energético de las aplicaciones manteniendo la funcionalidad exigida por el usuario.

El resto del artículo se organiza de la siguiente forma. En la Sección 2 se explican los conceptos básicos de la arquitectura Android en el contexto de la reconfiguración dinámica de las aplicaciones, y se discuten las propuestas existentes de reconfiguración aplicadas a dispositivos móviles, comparándolas con nuestra propuesta. La Sección 3 muestra el modelo genérico de adaptación propuesto, mientras que la Sección 4 detalla los mecanismos de reconfiguración propuestos que implementan el modelo de adaptación. La Sección 5 evalúa nuestra propuesta, calculando el consumo energético de los mecanismos de reconfiguración, así como el beneficio obtenido con la reconfiguración. Finalmente, la Sección 6 concluye el artículo y presenta el trabajo futuro.

2. Antecedentes y Trabajo Relacionado

Esta sección presenta los conocimientos necesarios sobre las aplicaciones Android en el contexto de la reconfiguración dinámica de las mismas y se discuten las propuestas existentes de reconfiguración, y en particular, aquellas basadas en la eficiencia energética, comparándolas con nuestra propuesta de adaptación.

2.1. Arquitectura Android

Existen dos entornos de ejecución de aplicaciones utilizados por el sistema operativo móvil Android dependiendo de la versión del sistema: (1) la máquina

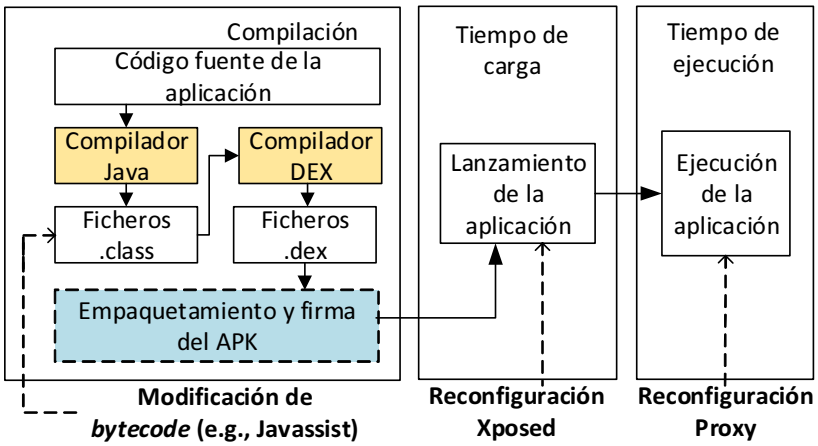

Figura 1. Proceso de compilación, lanzamiento y ejecución de una aplicación Android.

virtual *Dalvik*, utilizada originalmente en las primeras versiones de Android; y (2) el entorno de ejecución *ART* (*Android Runtime*) que reemplaza a Dalvik a partir de las versión 5.0 de Android. La diferencia principal entre ambos entornos radica en la compilación de aplicaciones. En el caso de *Dalvik*, el proceso de compilación es *Just-In-Time*: los ficheros de las aplicaciones se compilan a código máquina en el momento en el que se ejecutan. En ART, el proceso de compilación es *Ahead-Of-Time*, donde las aplicaciones se compilan a *bytecode* en el momento en el que se instalan en el dispositivo.

La Figura 1 resume el proceso de compilación, lanzamiento y ejecución de una aplicación Android. El compilador Java compila el código fuente de la aplicación, obteniendo el bytecode de Java (ficheros `.class`). Posteriormente, el compilador propio de Android (*DEX*) compila los ficheros `.class`, obteniendo el bytecode de Android (ficheros `.dex`). Los ficheros `.dex` se empaquetan y firman para construir la aplicación final *APK* lista para ser ejecutada.

2.2. Reconfiguración dinámica de código en Android

Teniendo en cuenta el proceso de compilación y ejecución de las aplicaciones Android (Figura 1), existen varias formas de reconfigurar dinámicamente una aplicación Android. En esta sección se presentan tres posibilidades en función de cómo se realice la modificación de la aplicación.

Modificación de la funcionalidad a nivel de *bytecode*. Actualmente, la modificación viable del código binario sólo es posible para el bytecode de Java, por medio de librerías como Javassist[1]. Librerías similares para el bytecode de Android no existen o están obsoletas como es el caso de Javassist-Android[2] o DX,[3] debido a la continua evolución y rápidas actualizaciones del sistema

[1] `http://jboss-javassist.github.io/javassist/`
[2] `https://github.com/crimsonwoods/javassist-android`
[3] `https://android.googlesource.com/platform/dalvik/+/refs/heads/jb-release/dx/`

Android. Sin embargo, la modificación del bytecode de Java no es un solución adecuada para reconfigurar dinámicamente una aplicación Android. Como se muestra en la Figura 1, esto se debe a que la modificación afecta a los ficheros `.class` antes de realizar la compilación con *DEX* y empaquetar la aplicación, por lo que la modificación sería realmente estática, en tiempo de compilación.

Reconfiguración usando el patrón de diseño *Proxy*. El patrón *Proxy* [6] permite controlar el acceso a un objeto. Definiendo *proxies* sobre las clases de la aplicación es posible controlar y delegar las llamadas que se realicen a la funcionalidad de esas clases, modificando el comportamiento de la aplicación en tiempo de ejecución (Figura 1).

Inyección de código: el framework *Xposed*. *Xposed*[4] es un framework que permite la instalación de módulos para controlar diversos aspectos de un dispositivo Android. Mediante su uso, se puede adaptar el comportamiento de una aplicación sin modificar su *APK*. Su funcionamiento se basa en crear una copia del proceso encargado de lanzar las aplicaciones, pudiendo modificar o incorporar nuevas clases a la aplicación justo antes de su ejecución, es decir, la adaptación se realiza en tiempo de carga (Figura 1).

2.3. Trabajo relacionado

Existen varios trabajos que, al igual que nuestra propuesta, utilizan reconfiguración dinámica de código para reducir el consumo energético de las aplicaciones [1,2,3,5,9,10]. La Tabla 2.3 compara los puntos más relevantes de cada propuesta comparándolas con nuestra solución.

En [1], Ardito define un módulo que notifica a las aplicaciones del consumo energético del hardware del dispositivo, enviando esta información a las aplicaciones suscritas que pueden reconfigurarse. El comportamiento de la aplicación depende del perfil de energía que se haya elegido (bajo consumo, consumo medio, etc). Aunque Ardito habla de eficiencia energética, no utiliza ninguna herramienta hardware ni software de medición de consumo, y los resultados obtenidos aluden al tiempo de duración total de la batería del dispositivo. Hay que tener que cuenta que en los dispositivos móviles se producen multitud de eventos simultáneos, y el porcentaje de batería al inicio y al final de una prueba no determina el consumo real de una aplicación. Además, las pruebas realizadas en [1] no aíslan la ejecución de la aplicación del resto del sistema. En [5], Elmalaki et al. define un framework (*CAreDroid*) para el desarrollo de aplicaciones cuyo funcionamiento depende del contexto del dispositivo. En CAreDroid, se define un conjunto de reglas en un fichero XML que determinan la funcionalidad que se ejecuta en cada momento en función del contexto en el que se encuentra el dispositivo (posición GPS, estado de la batería, etc). Al contrario que nuestra propuesta, CAreDroid está diseñado únicamente para smartphones Android con máquina virtual Dalvik. En [9,10] Gustavo et al. aplican algoritmos evolutivos

[4] `http://repo.xposed.info/`

Tabla 1. Comparativa de los trabajos relacionados y nuestra propuesta.

Característica	Ashmore [2] (*IMISSAR*)	Ardito [1]	Elmalaki [5] (*CAreDroid*)	Gustavo [9,10]	Casquina [3] (*Cosmapek*)	**Nuestra propuesta**
Tiene en cuenta la energía	□	■	■	■	□	■
Se ajusta al comportamiento del usuario	□	□	□	□	□	■
Carga de funcionalidad externa	■	□	□	□	□	■
Reconfiguración de la interfaz gráfica	□	□	□	□	■	■
Trasparente al usuario	□	□	■	■	■	■
Nivel de intrusión	Alto	Alto	Bajo	Bajo	Bajo	Bajo
Método/s de reconfiguración	DS	LI	LI	CD	LI	LI,IC

■ Característica soportada. □ Característica no soportada. LI: Lista de implementaciones de métodos y/o interfaces. DS: Delegación de funciones a servidor. CD: Configuración del dispositivo. IC: Inyección de código.

multiobjetivo para determinar la reconfiguración más adecuada en cada caso en función de unos valores de utilidad definidos previamente en base al contexto. La reconfiguración dinámica modifica la configuración del dispositivo, tales como la conectividad de red, el estado del Bluetooth o la calidad del sonido, pero no se realiza un cambio de código en las aplicaciones.

Por otro lado, existen propuestas similares de reconfiguración de código en dispositivos móviles pero sin el objetivo de mejorar la eficiencia energética de las aplicaciones [2,3]. En [2] Ashmore et al. definen *IMISSAR*, una plataforma que permite delegar el cómputo de los dispositivos móviles a un servidor, donde se prepara una máquina virtual del dispositivo. Los autores relacionan el coste de la reconfiguración con el beneficio obtenido, pero sin tener en cuenta el coste energético de la reconfiguración (sólo el nivel de batería). Además, esta solución requiere que el móvil esté continuamente conectado a Internet. En [3], Casquina et al. proponen *Cosmapek*, un mecanismo de reconfiguración que permite adaptar aplicaciones Android al contexto, basado en el modelo *MAPE-K loop*, por lo que presenta módulos cuya funcionalidad es parecida a nuestra propuesta. Su objetivo es adaptar la arquitectura en respuesta a posibles fallos que puedan ocurrir debido al contexto en el que se encuentre la aplicación. Sin embargo, ninguna de estas propuestas tiene en cuenta el consumo energético, ni el efecto del mecanismo de reconfiguración en las aplicaciones.

3. Modelo de Adaptación Eco-Eficiente

Para realizar la reconfiguración dinámica de código se propone un modelo de adaptación genérico (Figura 2). El modelo separa la aplicación base (`Aplicación-Android`) del sistema de reconfiguración, de forma que la adaptación se realiza de la manera menos intrusiva posible con respecto al código original. Los mó-

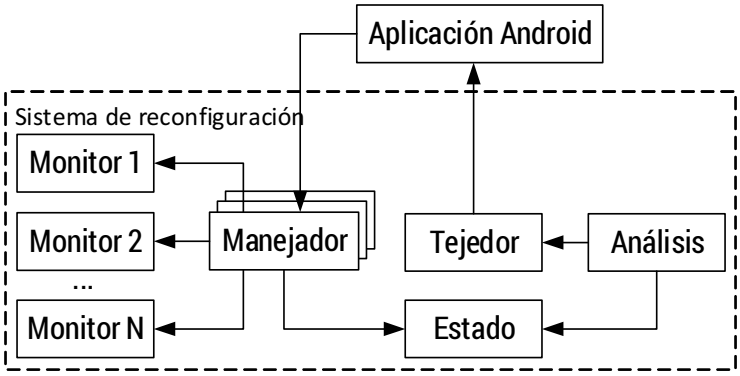

Figura 2. Modelo de adaptación genérico para aplicaciones Android.

dulos que componen el sistema de reconfiguración actúan sobre la aplicación, monitorizando y modificando el comportamiento de la misma:

- **Manejador.** Los manejadores (*handlers*) gestionan la funcionalidad de la aplicación que puede modificarse dinámicamente. Los manejadores interceptan las llamadas a las clases reconfigurables.
- **Monitor.** Los monitores controlan el estado del contexto de la aplicación. En particular, cada monitor se encarga de gestionar un conjunto de variables concreto de la aplicación.
- **Análisis.** El módulo de análisis toma una decisión acerca de la funcionalidad más adecuada que debe tener la aplicación en función del contexto en el que se encuentra. Para ello, se define un conjunto de reglas de reconfiguración y se utiliza la información del contexto proporcionada por los monitores para determinar la funcionalidad más apropriada.
- **Tejedor.** El tejedor (*Weaver*) es el módulo encargado de modificar directamente la funcionalidad (clases) de la aplicación, según la información proporcionada por el módulo de análisis. La implementación de este módulo determinará el mecanismo de reconfiguración específico que se usará (e.g., *proxies* dinámicos, *Xposed*).
- **Estado.** Este módulo se encarga de mantener el estado del contexto de la aplicación, incluso entre diferentes ejecuciones. Esto es, mantiene la información del contexto de uso de la aplicación de manera permanente a lo largo del tiempo.

4. Mecanismos de Reconfiguración Dinámicos

Esta sección presenta cuatro implementaciones diferentes del modelo de adaptación propuesto en la Sección 3 que se basan en el uso del patrón *Proxy* y/o el framework *Xposed* introducidos en la Sección 2 para modificar el comportamiento de las aplicaciones Android: (1) MR1: Proxy Interno; (2) MR2: Proxy Externo; (3) MR3: Xposed Proxy; y (4) MR4: Xposed.

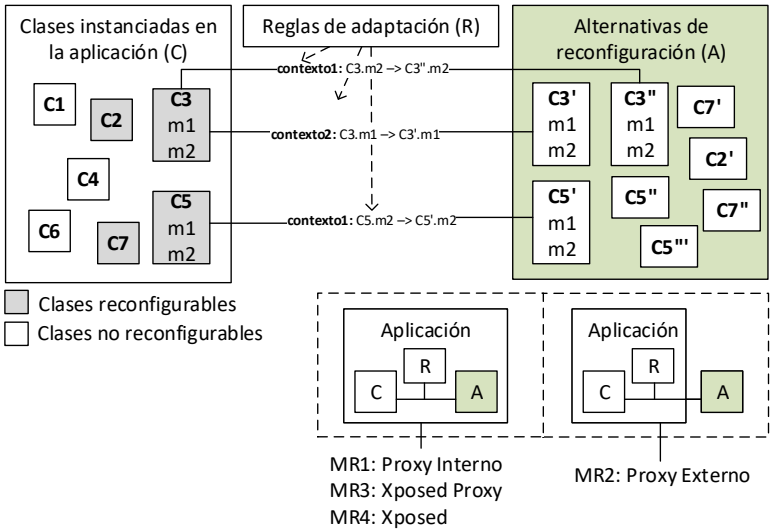

Figura 3. Esquema de reconfiguración basado en el patrón *Proxy*.

La Figura 3 muestra el esquema de reconfiguración que implementan los cuatro mecanismos de reconfiguración. La funcionalidad de la aplicación consta de un conjunto de clases instanciadas C que se divide en dos subconjuntos: un conjunto de clases fijas (no reconfigurables) y un conjunto de clases cuya funcionalidad (métodos) puede modificarse (clases reconfigurables, sombreadas en la Figura 3). Por otro lado, se dispone de un repositorio de clases A (Alternativas de reconfiguración) que proporcionan implementaciones alternativas a la funcionalidad (métodos) de las clases reconfigurables. Esto significa que las clases del conjunto A (e.g., la clase `C3'`) implementan las mismas interfaces que las clases reconfigurables de C (e.g., la clase `C3`). Finalmente, el conjunto de reglas de adaptación R indica qué funcionalidad o métodos (e.g., el método `m2` de la clase `C3`) se deben cambiar por la nueva funcionalidad o métodos (e.g., el método `m1` de la clase `C3''`), bajo un determinado contexto de la aplicación (e.g., `contexto1`).

La forma en la que se gestionan las clases reconfigurables y las clases alternativas de reconfiguración define el tipo de mecanismo de reconfiguración.

4.1. Mecanismos basados en el patrón Proxy

En este tipo de mecanismos, las clases reconfigurables implementan el patrón Proxy, que funciona a modo de manejador para las clases, controlando el acceso a la funcionalidad y los métodos de la clase. Cuando hay un cambio de contexto, el tejedor cambia la funcionalidad de la clase por la alternativa más eficiente energéticamente. En función de como estén disponibles las clases alternativas de reconfiguración, tendremos dos mecanismos de reconfiguración diferentes:

MR1: Proxy Interno. Las clases alternativas de reconfiguración se encuentran disponibles junto con el código de la aplicación listas para ser instan-

ciadas cuando se necesiten. Si alguna de estas clases no se utiliza durante la ejecución de la aplicación, no llegará a instanciarse, y por lo tanto, no consumirá recursos.

MR 2: Proxy Externo. La aplicación contiene únicamente la funcionalidad base, mientras que las clases alternativas de reconfiguración están contenidas en ficheros externos a la aplicación. Estos ficheros serán cargados bajo demanda, por lo que no consumirán recursos hasta el momento en el que el fichero de clases se cargue.

En ambos casos, la reconfiguración de código se produce en tiempo de ejecución, y de forma transparente al usuario.

4.2. Mecanismos basados en el *framework Xposed*

Como se detalla en la Sección 2, el framework Xposed permite modificar la aplicación justo antes de su ejecución. Al inicio de la aplicación, según el estado del dispositivo y el comportamiento del usuario en ejecuciones previas, y teniendo en cuenta las reglas de adaptación, se realiza la modificación de las clases reconfigurables por una funcionalidad de las alternativas de reconfiguración. En este caso, las alternativas de reconfiguración están disponibles durante el lanzamiento de la aplicación.

El cambio de la funcionalidad de las clases reconfigurables lo lleva a cabo directamente *Xposed*. Sin embargo, en función de como se controle el acceso a las clases reconfigurables para monitorizar el comportamiento del usuario, tendremos dos mecanismos de reconfiguración diferentes:

MR3: Xposed Proxy. El control del acceso y monitorización de las clases reconfigurables se realiza mediante el patrón Proxy. Sin embargo, a diferencia de los mecanismos MR1 y MR2 basados también en Proxy, esta solución no permite la modificación de las clases reconfigurables durante la ejecución de la aplicación.

MR 4: Xposed. En este caso, el control del acceso y monitorización de las clases reconfigurables se lleva a cabo mediante un sistema específico de intercepción de llamadas.

En ambos casos, la ventaja de *Xposed* sobre las implementaciones anteriores es que no es necesario gestionar el cambio de las clases reconfigurables ni las alternativas en tiempo de ejecución. Sólo es necesario monitorizar el comportamiento del usuario en tiempo de ejecución.

5. Evaluación

En esta sección evaluamos nuestra propuesta, realizando mediciones sobre el consumo energético de las diferentes soluciones propuestas bajo diferentes escenarios, con el objetivo de contestar a las siguientes preguntas de investigación:

RQ1: ¿Se puede reducir el consumo energético de las aplicaciones reconfigurándolas en base al comportamiento del usuario?

Tabla 2. Especificaciones de los dispositivos móviles usados en la experimentación.

Características	Móvil 1	Móvil 2
Modelo	Samsung Galaxy Nexus GT-i9250	LG Nexus 5 D821
Sistema	Android 4.3 Jelly Bean	Android 6.0 Marshmallow
Máquina virtual	Dalvik	ART
Arquitectura	32 bits, Texas Instruments MAP 4460 (45 nm)	32 bits, Qualcomm Snapdragon 800 MSM8974 (28 nm)
CPU	1.2 GHz Dual-Core ARM Cortex-A9	2.3 GHz Quad-Core Krait 400
GPU	PowerVR SGX540	Qualcomm Adreno 330
Memoria RAM	1 GB LPDDR2	2 GB LPDDR2
Memoria interna	16 GB	16 GB
Pantalla	4.65", 720x1280 px, 316 PPI	4.95", 1080x1920 px, 445 PPI
Conectividad	GSM Quad-Band, 2G, 3G, GPRS, EDGE, UMTS, HSDPA, HSUPA, HSPA+, WiFi 802.11 a/b/g/n, Bluetooth 3.0	GSM Quad-Band, 2G, 3G, 4G, GPRS, EDGE, UMTS, HSDPA, HSUPA, HSPA+, LTE, WiFi 802.11 a/b/g/n/ac+ MIMO, Bluetooth 4.0
Batería	Iones de Litio, 1750 mAh	Polímeros de Litio (LiPo), 2300 mAh

RQ2: ¿Los mecanismos de reconfiguración introducen un consumo de energía adicional en la aplicación y ese consumo es mayor que el beneficio obtenido de la reconfiguración?

RQ3: ¿Qué escenarios podrían aumentar el consumo energético de los mecanismos de reconfiguración propuestos?

Para evaluar la propuesta se ha desarrollado una aplicación base a modo de caso de estudio (Sección 5.1) y se han desarrollado las cuatro soluciones diferentes de adaptación propuestas en la Sección 4.[5]

5.1. Experimentación

Los experimentos se han llevado a cabo sobre dos dispositivos móviles diferentes y se han utilizado dos herramientas diferentes de medición: GreenOracle [4] y Trepn Power Profiler[6].

Dispositivos móviles. Se han seleccionado dos teléfonos móviles completamente diferentes (ver Tabla 2): (1) Samsung Galaxy Nexus con el sistema Android 4.3 y máquina virtual Dalvik; y (2) LG Nexus 5 con sistema Android 6.0 y máquina virtual ART. La diferencia en especificaciones de ambos teléfonos permite evaluar la aplicabilidad de nuestra propuesta.

Perfil energético de medición. La Figura 4 muestra el perfil energético de medición para los experimentos usando GreenOracle y Trepn Profiler. El modelo de energía propuesto por GreenOracle recoge un conjunto de estadísticas del sistema (e.g., llamadas al sistema, información de los tonos de la pantalla, duración de la prueba,...) y de la aplicación (e.g., uso de la CPU, uso de la red,...) y calcula una estimación del consumo de energía (en Julios) de la aplicación. Por otro lado, el modelo de energía de Trepn Profiler mide el consumo del sistema en general. Para estimar el consumo energético de una aplicación en concreto

[5] Código fuente disponible en `http://150.214.108.91/weaver-code/`.

[6] `https://developer.qualcomm.com/software/trepn-power-profiler`

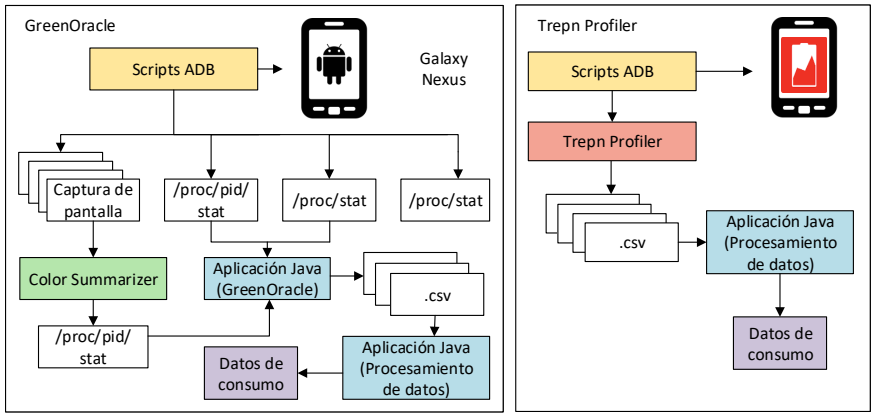

Figura 4. Esquemas de funcionamiento de GreenOracle y Trepn Profiler.

se utiliza un valor "delta" con el consumo del sistema antes y después de lanzar la aplicación. La Figura 4 muestra el esquema de funcionamiento de las herramientas de medición empleadas. Aunque éstas herramientas permiten analizar los aspectos de la aplicación qué más consumen (e.g., la pantalla y la GPU suelen consumir más energía que la CPU o la red en los dispositivos móviles [8]), nuestra propuesta valora el consumo energético global de aplicación.

Para cada experimento se han realizado 20 ejecuciones, obteniendo la media y desviación típica de los valores de consumo energético. Los experimentos se han ejecutado con el dispositivo en modo avión pero teniendo activado el receptor WiFi para enviar las instrucciones a través de la herramienta *ADB* (*Android Debug Bridge*). En los datos de consumo energético se incluye el consumo del sistema operativo. Los experimentos se han ejecutado para todos los mecanismos de reconfiguración con los dispositivos en las mismas condiciones.

Caso de estudio. Nuestra propuesta se ha evaluado con un caso de estudio basado en una aplicación de consulta de información. Por defecto, la aplicación obtiene, desde un servidor, información sobre resultados de fútbol de las ligas de 25 países. El consumo de la aplicación se puede mejorar modificando su comportamiento para que muestre sólo la información de las ligas que el usuario está realmente interesado, disminuyendo el número de peticiones al servidor y la cantidad de información a mostrar por la aplicación. Para ello, usando los mecanismos de reconfiguración propuestos, la funcionalidad de la aplicación se adapta al comportamiento del usuario. En particular, esta aplicación se adapta para realizar peticiones y mostrar información sólo para las ligas que el usuario ha consultado anteriormente tras un periodo de tiempo.

5.2. Resultados

En esta sección se muestra los resultados de los experimentos contestando a las preguntas de investigación planteadas.

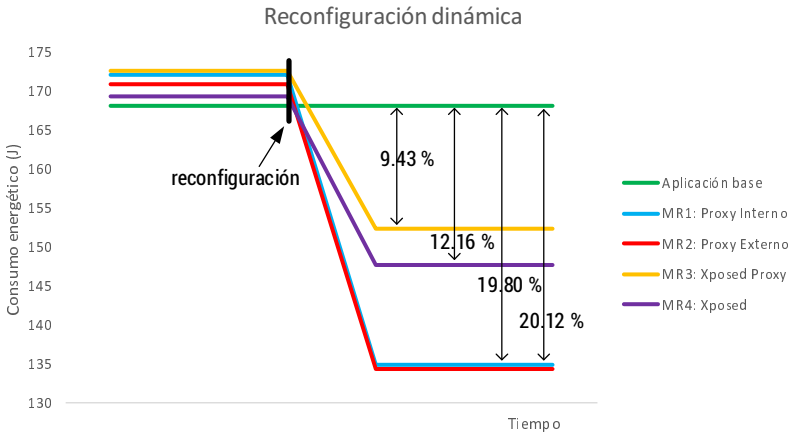

Figura 5. Consumo energético antes y después de la reconfiguración.

Beneficios de nuestra propuesta de reconfiguración. Para responder la pregunta RQ1, se ha medido el consumo energético de la aplicación base sin ningún tipo de mecanismo de reconfiguración bajo un determinado contexto, y a continuación se ha repetido el experimento reconfigurando la aplicación al cambio de contexto con los diferentes mecanismos propuestos.

La Figura 5 muestra el consumo antes y después de realizar la reconfiguración. Se puede observar un beneficio de hasta el 20 % usando la solución de reconfiguración basada en Proxy Internos (MR1) y Externos (MR2). Mientras que con las otras soluciones el beneficio es menor debido a que la adaptación se produce en el momento en que se lanza la aplicación, en lugar de en tiempo de ejecución. En los métodos MR1 y MR2 la aplicación se encuentra adaptada al contexto en todo momento.

Coste energético de los mecanismos de reconfiguración. Para calcular el coste energético de los mecanismos de reconfiguración (preguntas RQ1 y RQ2), (1) se han integrado las diferentes soluciones junto con la aplicación base; (2) se ha medido el consumo energético de la aplicación sin realizar ningún tipo de reconfiguración para observar el efecto de la funcionalidad adicional en el consumo energético; y (3) se ha medido el consumo de la aplicación reconfigurándola en función de diversos parámetros (número de clases reconfigurables, número de alternativas de reconfiguración, y número de reglas de adaptación).

En la Figura 6 se observa que el coste energético de las diferentes soluciones es mínimo, siendo un 2.68 % mayor con respecto al consumo de la aplicación base en el peor de los casos. Por lo que podemos afirmar que el consumo energético adicional que introducen los mecanismos de reconfiguración es insignificante con respecto a la mejora de consumo en la aplicación reconfigurada.

Para contestar a la pregunta RQ3, es necesario estudiar con mayor detalle diferentes escenarios de reconfiguración que pueden afectar al consumo energético. Estos escenarios solo tienen sentido para los mecanismos basados en Proxy Interno y Externo, debido a que la reconfiguración es totalmente dinámica en

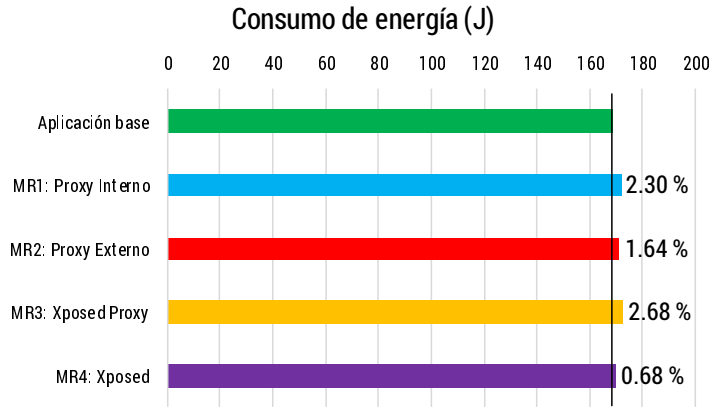

Figura 6. Consumo energético de los mecanismos de reconfiguración propuestos.

Figura 7. Consumo energético en relación con el número de clases reconfigurables.

tiempo de ejecución. Para los mecanismos basados en el framework *Xposed* no tiene sentido el siguiente análisis porque la modificación se realiza en tiempo de carga y por lo tanto no supone un coste de energía adicional en tiempo de ejecución.

Número de clases reconfigurables. La Figura 7 muestra el resultado de incrementar hasta 40 el número de clases reconfigurables instanciadas y gestionadas en la aplicación. Para la realización de la prueba, se ha incrementado el número de clases y de monitores en relación 1 a 1, por lo que, en el caso de 40 clases manejadas, coexisten 40 instancias de monitores para controlar el comportamiento del usuario. Se observa que el consumo energético de la aplicación no se ve incrementado, pese a haber un número importante de objetos dinámicos en la aplicación móvil.

Número de alternativas de reconfiguración. El número de alternativas de reconfiguración tampoco afecta al consumo energético de una aplicación normal como se observa en la Figura 8. En el caso de la solución Proxy Externo (MR2), cargar y gestionar hasta 100 clases diferentes no supone ningún coste energético

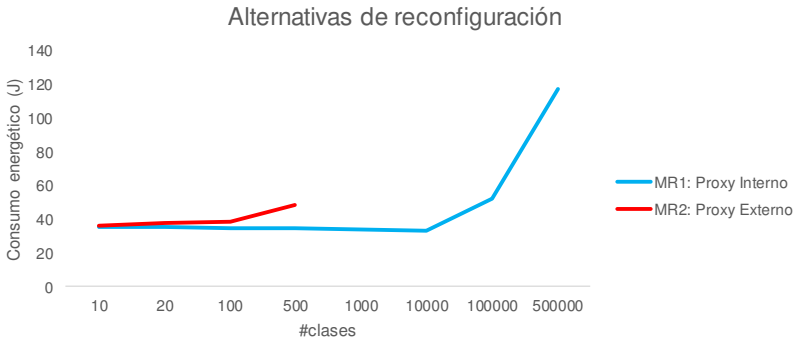

Figura 8. Consumo energético en función de las alternativas de reconfiguración.

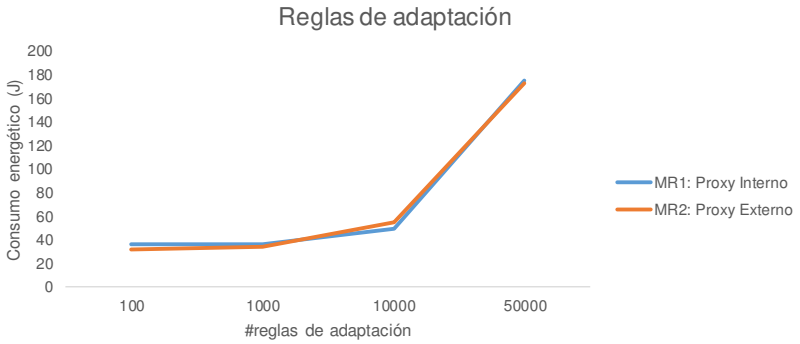

Figura 9. Consumo energético en relación con el número de reglas de reconfiguración.

adicional, incrementándose levemente a partir de 100. El número de clases a cargar externamente estaba limitado a 500 por las prestaciones de los dispositivos. En el caso de la solución Proxy Interno (MR1), llegamos incluso a las 10,000 clases sin que se vea afectado el consumo.

Número de reglas de adaptación. La Figura 9 muestra el consumo energético de las soluciones basadas en Proxy Interno (MR1) y Proxy Externo (MR2) en relación con el número de reglas de adaptación que contienen. Se observa que, hasta 10,000 reglas de adaptación, el sistema apenas aumenta su consumo. Este número de reglas de adaptación resulta inmanejable en una aplicación real, por lo que podemos concluir que el número de reglas de adaptación no interviene en el consumo energético de la reconfiguración.

En un aplicación móvil real, el número de clases reconfigurables y de reglas de adaptación serán muy inferiores a los considerados en los experimentos. Podemos concluir que el número de clases reconfigurables y la cantidad de reglas de adaptación no influye en el consumo de la reconfiguración en aplicaciones reales.

6. Conclusiones y Trabajo Futuro

En este artículo se ha presentado un modelo de adaptación dinámico para aplicaciones móviles con el objetivo de reducir su consumo energético. La adapta-

ción se realiza en función del comportamiento del usuario y del contexto y estado de la aplicación. A su vez, se proporcionan cuatro mecanismos de reconfiguración diferentes que implementan el modelo de adaptación propuesto.

Se ha demostrado que estos mecanismos de reconfiguración no suponen un consumo energético adicional en comparación con los beneficios de la reconfiguración. En particular, en nuestro caso de estudio, reconfigurar la aplicación en base al comportamiento del usuario mejora el consumo energético de la aplicación en un 20 %.

Como trabajo futuro, se plantea la evaluación del consumo energético de las propuestas de reconfiguración existentes cuyo coste energético se desconoce. Además, se estudiará la viabilidad energética de una propuesta de reconfiguración dinámica basada en la orientación a aspectos para dispositivos móviles.

Agradecimientos

Trabajo financiado por los proyectos MAGIC P12-TIC1814 y HADAS TIN2015-64841-R, y por la Universidad de Málaga.

Referencias

1. Ardito, L.: Energy aware self-adaptation in mobile systems. In: Software Engineering (ICSE), 2013 35th International Conference on. pp. 1435–1437. IEEE (2013)
2. Ashmore, S., Makki, S.K.: Imissar: an intelligent, mobile middleware solution for secure automatic reconfiguration of applications, utilizing a feature model approach. In: Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication. p. 58. ACM (2011)
3. Casquina, J.C., Eleuterio, J.D.S., Rubira, C.M.: Adaptive deployment infrastructure for android applications. In: Dependable Computing Conference (EDCC), 2016 12th European. pp. 218–228. IEEE (2016)
4. Chowdhury, S.A., Hindle, A.: Greenoracle: Estimating software energy consumption with energy measurement corpora. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). pp. 49–60 (May 2016)
5. Elmalaki, S., Wanner, L., Srivastava, M.: Caredroid: Adaptation framework for android context-aware applications. In: Proceedings of the 21st Annual International Conference on Mobile Computing and Networking. pp. 386–399. ACM (2015)
6. Gamma, E.: Design patterns: elements of reusable object-oriented software. Pearson Education India (1995)
7. Index, C.V.N.: Global mobile data traffic forecast update, 2016–2021 white paper, accessed on may 2, 2017
8. Pang, C., Hindle, A., Adams, B., Hassan, A.E.: What do programmers know about software energy consumption? IEEE Software 33(3), 83–89 (May 2016)
9. Pascual, G.G., Lopez-Herrejon, R.E., Pinto, M., Fuentes, L., Egyed, A.: Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. Journal of Systems and Software 103, 392–411 (2015)
10. Pascual, G.G., Pinto, M., Fuentes, L.: Self-adaptation of mobile systems driven by the common variability language. Future Generation Computer Systems 47, 127–144 (2015)